Article
Guillaume Rongier · Août 24, 2022 9m de lecture

Open Exchange

Découvrir Django, partie 1

Il y a quelque temps, j'ai présenté un nouveau pilote de Django pour IRIS. Maintenant, voyons comment utiliser Django avec IRIS en pratique.

Remarque importante : l'utilisation de Django avec IRIS Community Edition ne fonctionnera pratiquement pas, Community Edition n'a que 5 connexions disponibles, et Django les utilisera très rapidement. Malheureusement, pour cette raison, je ne peux pas recommander cette méthode pour le développement de nouvelles applications, en raison de la difficulté à prévoir l'utilisation des licences.

Lancement du projet Django

Tout d'abord, démarrons notre nouveau projet Django. Pour ce faire, nous devons d'abord installer Django lui-même.

```
pip install django
```

Ensuite, créez un projet nommé demo, cela permettra de créer un dossier de projets avec le même nom

```
django-admin startproject demo

cd demo
```

ou vous pouvez le faire dans un dossier existant

```
django-admin startproject main .
```

Cette commande va remplir quelques fichiers python pour nous.

Where,

- **manage.py:** utilitaire de ligne de commande qui vous permet d'interagir avec ce projet Django de diverses manières
- **main** le répertoire correspondant au paquetage Python de votre projet
- **main/init.py:** un fichier vide qui indique à Python que ce répertoire doit être considéré comme un package Python
- **main/settings.py**: paramètres/configuration pour ce projet Django
- **main/urls.py**: Les déclarations d'URL pour ce projet Django ; une "table des matières" de votre site alimenté par Django.
- **main/asci.py**: Un point d'entrée pour les serveurs web compatibles avec ASGI pour servir votre projet.
- **main/wsci.py**: Un point d'entrée pour les serveurs web compatibles avec WSGI pour servir votre projet.

Même à partir de ce point, nous pouvons commencer notre projet et il fonctionnera dans une certaine mesure

```
$ python manage.py runserver

Contrôle des modifications de fichiers avec StatReloader
Réalisation de vérifications du système...

La vérification du système n'a identifié aucun problème (0 désactivé). Vous avez 18 m
igration(s) non appliquée(s). Votre projet ne fonctionnera pas correctement tant que
vous n'aurez pas appliqué les migrations pour les applications suivantes : admin, aut
h, contenttypes, sessions.
Lancez 'python manage.py migrate' pour les appliquer.
22 juillet 2022 - 15:24:12
Django version 4.0.6, en utilisant les paramètres 'main.settings'.
Démarrage du serveur de développement à l'adresse http://127.0.0.1:8000/
Quittez le serveur avec CONTROL-C.
```

Now you can go to the browser and open URL http://127.0.0.1:8000 there

Ajout IRIS

Ajoutons l'accès à IRIS, et pour le faire nous devons installer quelques dépendances à notre projet, et la bonne façon de le faire, est de le définir dans un fichier nommé requirements.txt avec ce contenu, où nous devons ajouter django comme une dépendance_

```
# Django itself
django>=4.0.0
```

Et ensuite, le pilote pour IRIS pour Django, est publié. *Malheureusement, InterSystems ne veut pas publier ses propres pilotes sur PyPI, nous devons donc les définir de cette horrible façon. Sachez qu'ils peuvent le supprimer à tout moment, et qu'il peut donc ne plus fonctionner à l'avenir. (S'il était dans PyPI, il serait installé comme une dépendance de django-iris, et ne serait pas nécessaire de le définir explicitement)*

```
# Pilote IRIS d'InterSystems pour Django et pilote DB-API d'InterSystems
django-iris==0.1.13
https://raw.githubusercontent.com/intersystems-community/iris-driver-
distribution/main/DB-API/intersystems_irispython-3.2.0-py3-none-any.whl
```

Installez les dépendances définies dans ce fichier avec la commande

```
pip install -r requirements.txt
```

Maintenant, nous pouvons configurer notre projet pour utiliser IRIS, pour ce faire, nous devons mettre à jour le paramètre **DATABASES** dans le fichier *settings.py*, avec des lignes comme celle-ci, où NAME pointe vers l'espace de nom dans IRIS, et le port vers le SuperPort où IRIS est disponible

```
DATABASES = {
    'default': {
        'ENGINE': 'django_iris',
        'NAME': 'USER',
        'USER': '_SYSTEM',
        'PASSWORD': 'SYS',
        'HOST': 'localhost',
        'PORT': 1982,
    }
}
```

Django a un ORM, et des modèles stockés dans le projet, et cela nécessite de synchroniser les modèles Django avec la base de données en tant que tableaux. Par défaut, il y a quelques modèles liés à l'authentification. Et nous pouvons lancer le processus de migration maintenant

```
$ python manage.py migrate
Opérations à effectuer :
  Appliquer toutes les migrations : admin, auth, contenttypes, sessions
Exécution des migrations :
  Application de contenttypes.0001_initial... OK
  Application d'auth.0001_initial... OK
  Application d'admin.0001_initial... OK
  Application d'admin.0002_logentry_remove_auto_add... OK
  Application d'admin.0003_logentry_add_action_flag_choices... OK
  Application de contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Application d'auth.0003_alter_user_email_max_length... OK
  Application d'auth.0004_alter_user_username_opts... OK
  Application d'auth.0005_alter_user_last_login_null... OK
  Application d'auth.0006_require_contenttypes_0002... OK
  Application d'auth.0007_alter_validators_add_error_messages... OK
  Application d'auth.0008_alter_user_username_max_length... OK
  Application d'auth.0009_alter_user_last_name_max_length... OK
  Application d'auth.0010_alter_group_name_max_length... OK
  Application d'auth.0011_update_proxy_permissions... OK
  Application d'auth.0012_alter_user_first_name_max_length... OK
  Application de sessions.0001_initial... OK
```

Si vous allez à l'IRIS, vous y trouverez des tableaux supplémentaires

Définissez d'autres modèles

Il est temps d'ajouter certains de nos modèles. Pour ce faire, ajoutez un nouveau fichier *models.py,* avec un contenu comme suit

```
from django.db import models
```

```
class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    dob = models.DateField()
    sex = models.BooleanField()
```

Comme vous pouvez le voir, il comporte différents types de champs. Ensuite, ce modèle doit être préparé pour la base de données. Avant de le faire, ajoutez notre projet **main** à **INSTALLEDAPPS** dans *settings.py*

```
INSTALLED_APPS = [
    ....
    'main',
]
```

Et nous pouvons exécuter makemigrations. Cette commande doit être appelée après toute modification du modèle, elle s'occupe des modifications historiques du modèle, et quelle que soit la version de l'application installée, la migration saura mettre à jour le schéma de la base de données

```
$ python manage.py makemigrations main
Migrations pour 'main':
  main/migrations/0001_initial.py
    - Créez le modèle Person
```

Nous pouvons exécuter migrate à nouveau, il sait déjà que les migrations précédentes ont déjà été effectuées, donc, il exécute seulement la nouvelle migration

```
$ python manage.py migrate
Opérations à effectuer :
  Application de toutes les migrations : admin, auth, contenttypes, main, sessions
Exécution des migrations :
  Application de main.0001_initial... OK
```

Et actuellement, nous pouvons voir comment la migration se présente du point de vue de SQL

```
$ python manage.py sqlmigrate main 0001
--
-- Créez le modèle Person
--
CREAEZ TABLEAU "main_person" ("id" BIGINT AUTO_INCREMENT NOT NULL PRIMARY KEY, "first
_name" VARCHAR(30) NULL, "last_name" VARCHAR(30) NULL, "dob" DATE NOT NULL, "sex" BIT
 NOT NULL);
```

Mais il est possible d'avoir accès aux tableaux déjà existants dans la base de données, par exemple, si vous avez déjà une application qui fonctionne. J'ai installé le paquet zpm posts-and-tags, faisons un modèle pour le tableau community.posts

```
$ python manage.py inspectdb community.post
# Ceci est un module de modèle Django généré automatiquement.
# Vous devrez faire ce qui suit manuellement pour nettoyer tout ceci :
# * Réorganiser l'ordre des modèles
# * S'assurer que chaque modèle a un champ avec primary_key=True
# * Assurez-vous que chaque ForeignKey, et OneToOneField a un `on_delete` réglé sur l
e comportement désiré.
```

```
# * Supprimez les lignes `managed = False` si vous souhaitez autoriser Django à créer
, modifier et supprimer le tableau.
# N'hésitez pas à renommer les modèles, mais ne renommez pas les valeurs de db_table
ou les noms de champs
à partir de django.db import models

classe CommunityPost(models.Model):
    id = models.AutoField(db_column='ID')  # Nom du champ en minuscules.
    acceptedanswerts = models.DateTimeField(db_column='AcceptedAnswerTS', blank=True,
 null=True)  # Nom du champ en minuscules.
    author = models.CharField(db_column='Author', max_length=50, blank=True, null=Tru
e)  # Nom du champ en minuscules.
    avgvote = models.IntegerField(db_column='AvgVote', blank=True, null=True)  # Nom
du champ en minuscules.
    commentsamount = models.IntegerField(db_column='CommentsAmount', blank=True, null
=True)  # Nom du champ en minuscules.
    created = models.DateTimeField(db_column='Created', blank=True, null=True)  # Nom
 du champ en minuscules.
    deleted = models.BooleanField(db_column='Deleted', blank=True, null=True)  # Nom
du champ en minuscules.
    favscount = models.IntegerField(db_column='FavsCount', blank=True, null=True)  #
Nom du champ en minuscules.
    hascorrectanswer = models.BooleanField(db_column='HasCorrectAnswer', blank=True,
null=True)  # Nom du champ en minuscules.
    hash = models.CharField(db_column='Hash', max_length=50, blank=True, null=True)
# Nom du champ en minuscules.
    lang = models.CharField(db_column='Lang', max_length=50, blank=True, null=True)
# Nom du champ en minuscules.
    name = models.CharField(db_column='Name', max_length=250, blank=True, null=True)
 # Nom du champ en minuscules.
    nid = models.IntegerField(db_column='Nid', primary_key=True)  # Nom du champ en m
inuscules.
    posttype = models.CharField(db_column='PostType', max_length=50, blank=True, null
=True)  # Nom du champ en minuscules.
    published = models.BooleanField(db_column='Published', blank=True, null=True)  #
Nom du champ en minuscules.
    publisheddate = models.DateTimeField(db_column='PublishedDate', blank=True, null=
True)  # Nom du champ en minuscules.
    subscount = models.IntegerField(db_column='SubsCount', blank=True, null=True)  #
Nom du champ en minuscules.
    tags = models.CharField(db_column='Tags', max_length=350, blank=True, null=True)
 # Nom du champ en minuscules.
    text = models.CharField(db_column='Text', max_length=-1, blank=True, null=True)
# Nom du champ en minuscules.
    translated = models.BooleanField(db_column='Translated', blank=True, null=True)
# Nom du champ en minuscules.
    type = models.CharField(db_column='Type', max_length=50, blank=True, null=True)
# Nom du champ en minuscules.
    views = models.IntegerField(db_column='Views', blank=True, null=True)  # Nom du c
hamp en minuscules.
    votesamount = models.IntegerField(db_column='VotesAmount', blank=True, null=True)
  # Nom du champ en minuscules.

    classe Meta:
        managed = False
        db_table = 'community.post'
```

Elle est marquée comme **managed = False**, ce qui signifie que makemigrations et migrate ne fonctionneront pas pour cette table. En omettant le nom du tableau, vous obtiendrez une grande liste de modules, y compris les

tableaux déjà créés par Django auparavant

#Python #InterSystems IRIS
Voir l'application sur InterSystems Open Exchange