Artículo
Ricardo Paiva · 10 nov, 2022 Lectura de 8 min

Open Exchange

Introducción a Django - 1ª parte

Hace tiempo presenté un nuevo *driver* para Django for IRIS. Ahora voy a mostrar cómo utilizar Django con IRIS en la práctica.

Nota importante: Django no funciona bien con la Community Edition de IRIS. La Community Edition solo tiene disponibles 5 conexiones, y Django las usará muy rápido. Así que, desafortunadamente por esta razón no puedo recomendar este método para desarrollar nuevas aplicaciones, debido a la dificultad de predecir el uso de la licencia.

Inicio del proyecto Django

Para empezar, tenemos que instalar Django

```
pip install django
```

Después, creamos un proyecto llamado demo. Creará una carpeta de proyectos con el mismo nombre

```
django-admin startproject demo
```

```
cd demo
```

o se puede hacer en una carpeta existente

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

Este comando genera unos pocos archivos python para nosotros.

Donde:

- **manage.py:** utilidad de línea de comando que permite interactuar con este proyecto Django de varias maneras
- directorio **main**: es el paquete Python actual para vuestro proyecto
- **main/init.py:** un archivo vacío que dice a Python que este directorio debería ser consideredo un paquete Python
- **main/settings.py**: settings/configuración para este proyecto Django
- **main/urls.py**: Las declaraciones URL para este proyecto Django; un " índice " de vuestro sitio con la funcioanlidad de Django
- **main/asci.py**: Un punto de entrada para servidores web compatibles con ASGI para atender vuestro proyecto
- **main/wsci.py**: Un punto de entrada para servidores web compatibles con WSGI para atender vuestro proyecto

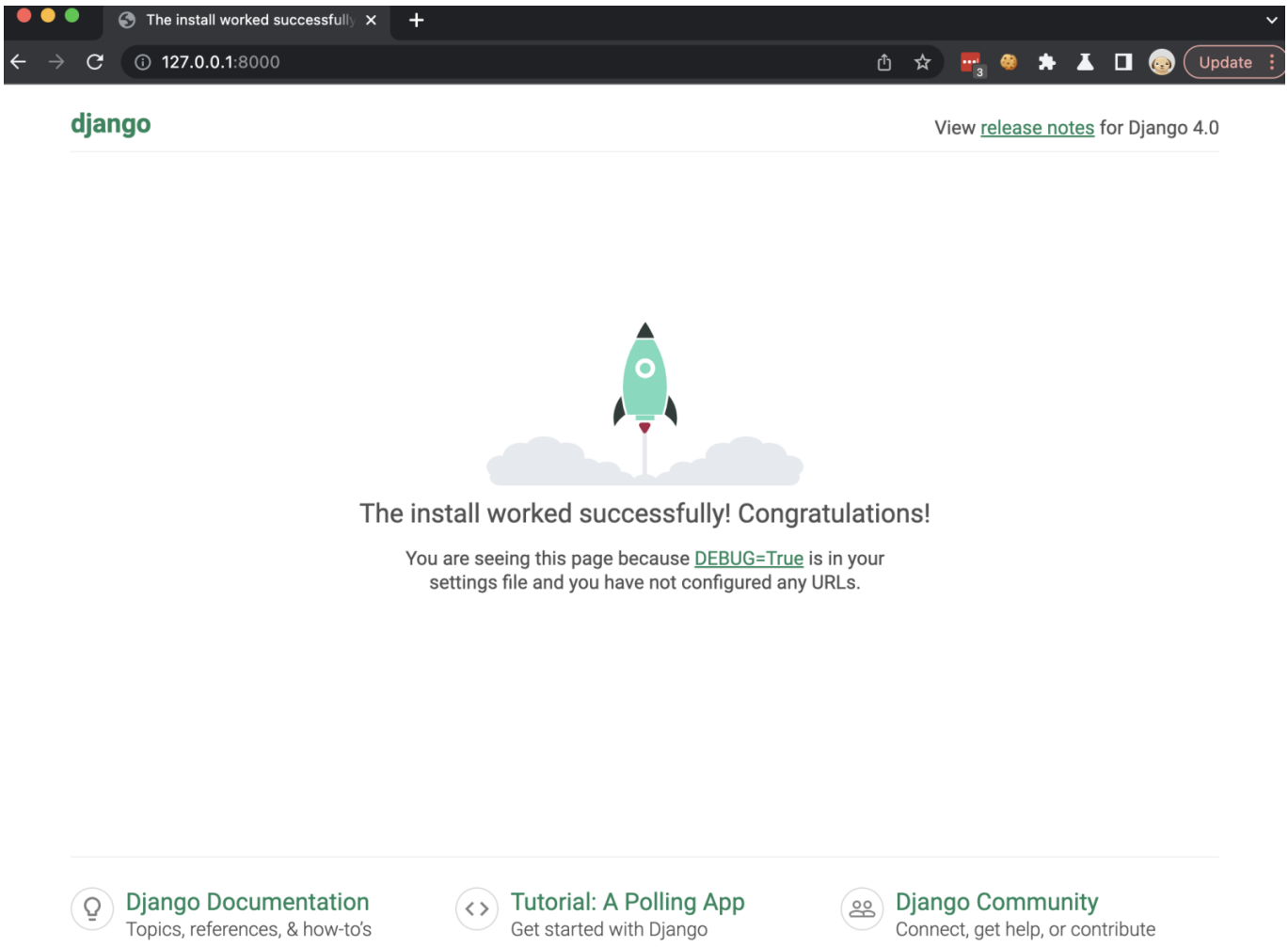
Incluso desde este punto podemos empezar nuestro proyecto y, de alguna manera, funcionará

```
$ python manage.py runserver

Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced). You have 18 unapplied migration(s). Y
our project may not work properly until you apply the migrations for app(s): admin, a
uth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
July 22, 2022 - 15:24:12
Django version 4.0.6, using settings 'main.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Ahora, se puede ir al navegador y abrir la URL http://127.0.0.1:8000 allí



Añadir IRIS

Vamos a añadir acceso a IRIS, y para hacerlo necesitamos instalar unas cuantas dependencias de nuestro proyecto, y la forma correcta de hacerlo es definirlo en el archivo llamado *requirements.txt* con este contenido, donde deberíamos añadir django como una dependencia

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

Y después, el *driver* de IRIS for Django, publicado. *Desafortunadamente, InterSystems no quiere publicar drivers propios en PyPI, así que tenemos que definirlo de esta manera más "fea". Estad atentos, pueden eliminarlo en cualquier momento, por lo que puede dejar de funcionar en el futuro. (Si estuviera en pypi, se instalaría como una dependencia de django-iris, no siendo necesaria su definición.)*

```
# InterSystems IRIS driver for Django, and DB-API driver from 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
```

Instalar dependencias definidas en este archivo con comando

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

Ahora, podemos configurar nuestro proyecto para usar IRIS. Para hacerlo, tenemos que actualizar el parámetro **DATABASES** en *el archivo settings.py*, con líneas como esta, en la que NAME apunta al Namespace en IRIS, y la puerta al SuperPort donde tenemos IRIS.

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

Django tiene ORM, y modelos almacenados en el proyecto, y requiere modelos de Django sincronizados con Database como tablas. Por defecto, hay unos pocos modelos, relacionados con auth. Y podemos ejecutar migrate ahora

```
$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
```

Si vamos a IRIS, encontraremos varias tablas extra allí

Catalog Details | Execute Query | **Browse** | SQL Statements

Show All Schemas **Show Schemas with Filter**

(10) Tables with schema: SQLUser

Name	Owner	Last Compiled	External	ReadOnly	Class Name	EXTENTSIZE
auth_group	_SYSTEM	2022-07-22 19:24:14	0	0	User.authgroup	100000
auth_group_permissions	_SYSTEM	2022-07-22 19:24:08	0	0	User.authgrouppermissions	100000
auth_permission	_SYSTEM	2022-07-22 19:24:15	0	0	User.authpermission	0
auth_user	_SYSTEM	2022-07-22 19:24:15	0	0	User.authuser	100000
auth_user_groups	_SYSTEM	2022-07-22 19:24:10	0	0	User.authusergroups	100000
auth_user_user_permissions	_SYSTEM	2022-07-22 19:24:11	0	0	User.authuseruserpermissions	100000
django_admin_log	_SYSTEM	2022-07-22 19:24:12	0	0	User.djangoadminlog	100000
django_content_type	_SYSTEM	2022-07-22 19:24:15	0	0	User.djangocontenttype	0
django_migrations	_SYSTEM	2022-07-22 19:24:05	0	0	User.djangomigrations	1
django_session	_SYSTEM	2022-07-22 19:24:15	0	0	User.djangosession	100000

Definir más modelos

Es el momento de añadir algunos de nuestros modelos. Para hacerlo, añade un nuevo archivo *models.py,* con contenido como este

```
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()
```

Como puedes ver, tiene diferentes tipos de campos. Entonces este modelo tiene que estar preparado para Database. Antes de hacerlo, añade nuestro proyecto **main** a **INSTALLEDAPPS** en *settings.py*

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

Y podemos makemigrations. Este comando tiene que ser llamado después de cualquier cambio en el modelo, se ocupa de los cambios históricos en el modelo, y no importa qué versión de la aplicación está instalada, la migración sabrá cómo actualizar el esquema en la base de datos

```
$ python manage.py makemigrations main
Migrations for 'main':
  main/migrations/0001_initial.py
    - Create model Person
```

Podemos ejecutar migrate otra vez, ya sabe que las migraciones anteriores ya están hechas, así que solo ejecuta la nueva

```
$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, main, sessions
```

```
Running migrations:
  Applying main.0001_initial... OK
```

Y, de hecho, podemos ver cómo se ve la migración desde la vista de SQL.

```
$ python manage.py sqlmigrate main 0001
--
-- Create model Person
--
CREATE TABLE "main_person" ("id" BIGINT AUTO_INCREMENT NOT NULL PRIMARY KEY, "first_n
ame" VARCHAR(30) NULL, "last_name" VARCHAR(30) NULL, "dob" DATE NOT NULL, "sex" BIT N
OT NULL);
```

Pero es posible acceder a las tablas ya existentes en la base de datos, por ejemplo, si ya tienes una aplicación funcionando. Yo tengo el gestor de paquetes zpm package posts-and-tags instalado, vamos a hacer un modelo para la tabla community.posts

```
$ python manage.py inspectdb community.post
# This is an auto-generated Django model module.
# You'll have to do the following manually to clean this up:
#   * Rearrange models' order
#   * Make sure each model has one field with primary_key=True
#   * Make sure each ForeignKey and OneToOneField has `on_delete` set to the desired
behavior
#   * Remove `managed = False` lines if you wish to allow Django to create, modify, a
nd delete the table
# Feel free to rename the models, but don't rename db_table values or field names.
from django.db import models

class CommunityPost(models.Model):
    id = models.AutoField(db_column='ID')  # Field name made lowercase.
    acceptedanswerts = models.DateTimeField(db_column='AcceptedAnswerTS', blank=True,
 null=True)  # Field name made lowercase.
    author = models.CharField(db_column='Author', max_length=50, blank=True, null=Tru
e)  # Field name made lowercase.
    avgvote = models.IntegerField(db_column='AvgVote', blank=True, null=True)  # Fiel
d name made lowercase.
    commentsamount = models.IntegerField(db_column='CommentsAmount', blank=True, null
=True)  # Field name made lowercase.
    created = models.DateTimeField(db_column='Created', blank=True, null=True)  # Fie
ld name made lowercase.
    deleted = models.BooleanField(db_column='Deleted', blank=True, null=True)  # Fiel
d name made lowercase.
    favscount = models.IntegerField(db_column='FavsCount', blank=True, null=True)  #
Field name made lowercase.
    hascorrectanswer = models.BooleanField(db_column='HasCorrectAnswer', blank=True,
null=True)  # Field name made lowercase.
    hash = models.CharField(db_column='Hash', max_length=50, blank=True, null=True)
# Field name made lowercase.
    lang = models.CharField(db_column='Lang', max_length=50, blank=True, null=True)
# Field name made lowercase.
    name = models.CharField(db_column='Name', max_length=250, blank=True, null=True)
 # Field name made lowercase.
    nid = models.IntegerField(db_column='Nid', primary_key=True)  # Field name made l
owercase.
    posttype = models.CharField(db_column='PostType', max_length=50, blank=True, null
=True)  # Field name made lowercase.
```

```
    published = models.BooleanField(db_column='Published', blank=True, null=True)  #
Field name made lowercase.
    publisheddate = models.DateTimeField(db_column='PublishedDate', blank=True, null=
True)  # Field name made lowercase.
    subscount = models.IntegerField(db_column='SubsCount', blank=True, null=True)  #
Field name made lowercase.
    tags = models.CharField(db_column='Tags', max_length=350, blank=True, null=True)
 # Field name made lowercase.
    text = models.CharField(db_column='Text', max_length=-1, blank=True, null=True)
# Field name made lowercase.
    translated = models.BooleanField(db_column='Translated', blank=True, null=True)
# Field name made lowercase.
    type = models.CharField(db_column='Type', max_length=50, blank=True, null=True)
# Field name made lowercase.
    views = models.IntegerField(db_column='Views', blank=True, null=True)  # Field na
me made lowercase.
    votesamount = models.IntegerField(db_column='VotesAmount', blank=True, null=True)
  # Field name made lowercase.

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

Está marcado como **managed = False**, lo que significa que makemigrations y migrate no funcionarán para esta tabla. Omitiendo el nombre de la tabla, producirá una lista larga de módulos, incluyendo las tablas ya creadas con Django previamente.

#Python #InterSystems IRIS
Ir a la aplicación en InterSystems Open Exchange

URL de fuente:https://es.community.intersystems.com/post/introducci%C3%B3n-django-1%C2%AA-parte