Article
Dmitry Maslennikov · Jul 22, 2022 8m read

Open Exchange

Introduction to Django part 1

Some time ago I introduced a new driver for Django for IRIS. Now, let's see how to use Django with IRIS in practice.

Important Note: Using Django with IRIS Community Edition will not work almost at all, Community Edition has only 5 connections available, and Django will use it very quickly. So, unfortunately by this reason I can't recommend this way for development any new applications, due to the difficulty of predicting license usage.

Start Django project

First of all let's start our new Django project, to do it, firstly, we have to install Django itself

```
pip install django
```

Then create a project named demo, it will create a projects folder with the same name

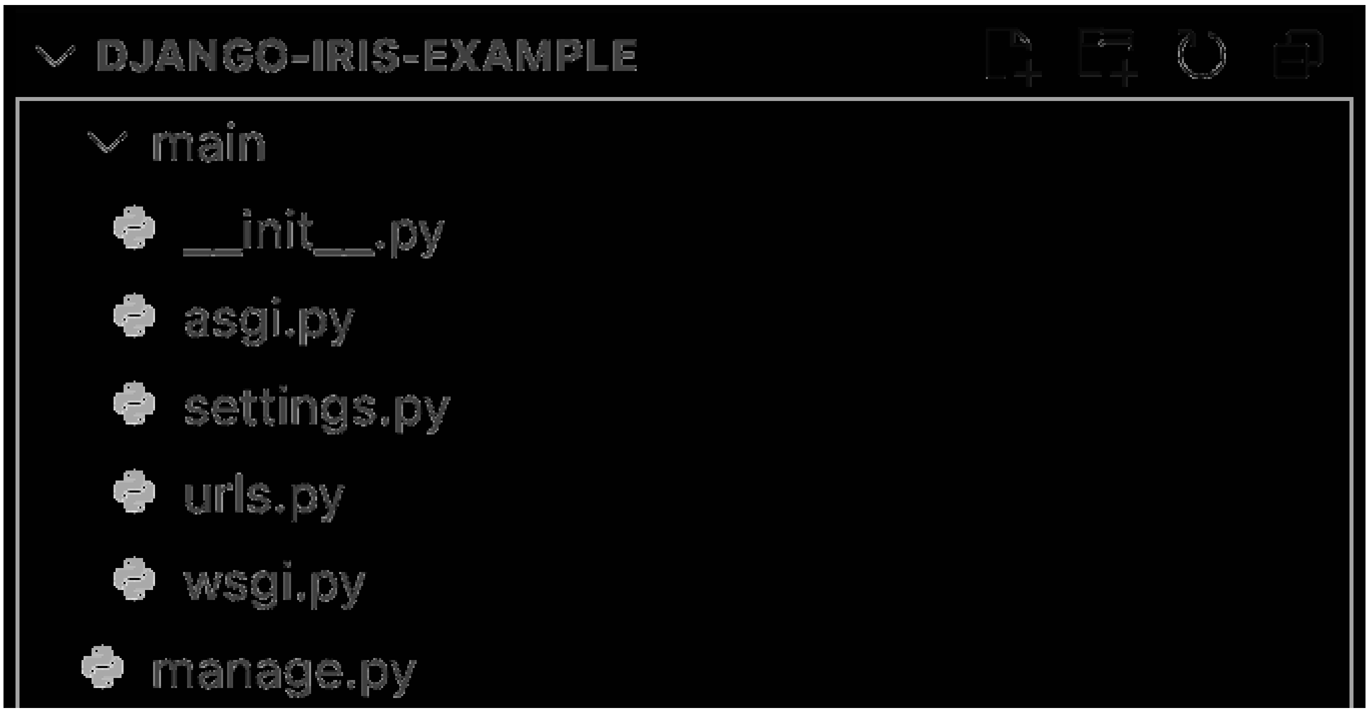
```
django-admin startproject demo

cd demo
```

or you may do it in an existing folder

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

This command will populate a few python files for us.



Where,

- **manage.py:** command-line utility that lets you interact with this Django project in various ways
- **main** directory: is the actual Python package for your project
- **main/init.py:** an empty file that tells Python that this directory should be considered a Python package
- **main/settings.py**: settings/configuration for this Django project
- **main/urls.py**: The URL declarations for this Django project; a " table of contents " of your Django-powered site.
- **main/asci.py**: An entry-point for ASGI-compatible web servers to serve your project.
- **main/wsci.py**: An entry-point for WSGI-compatible web servers to serve your project.

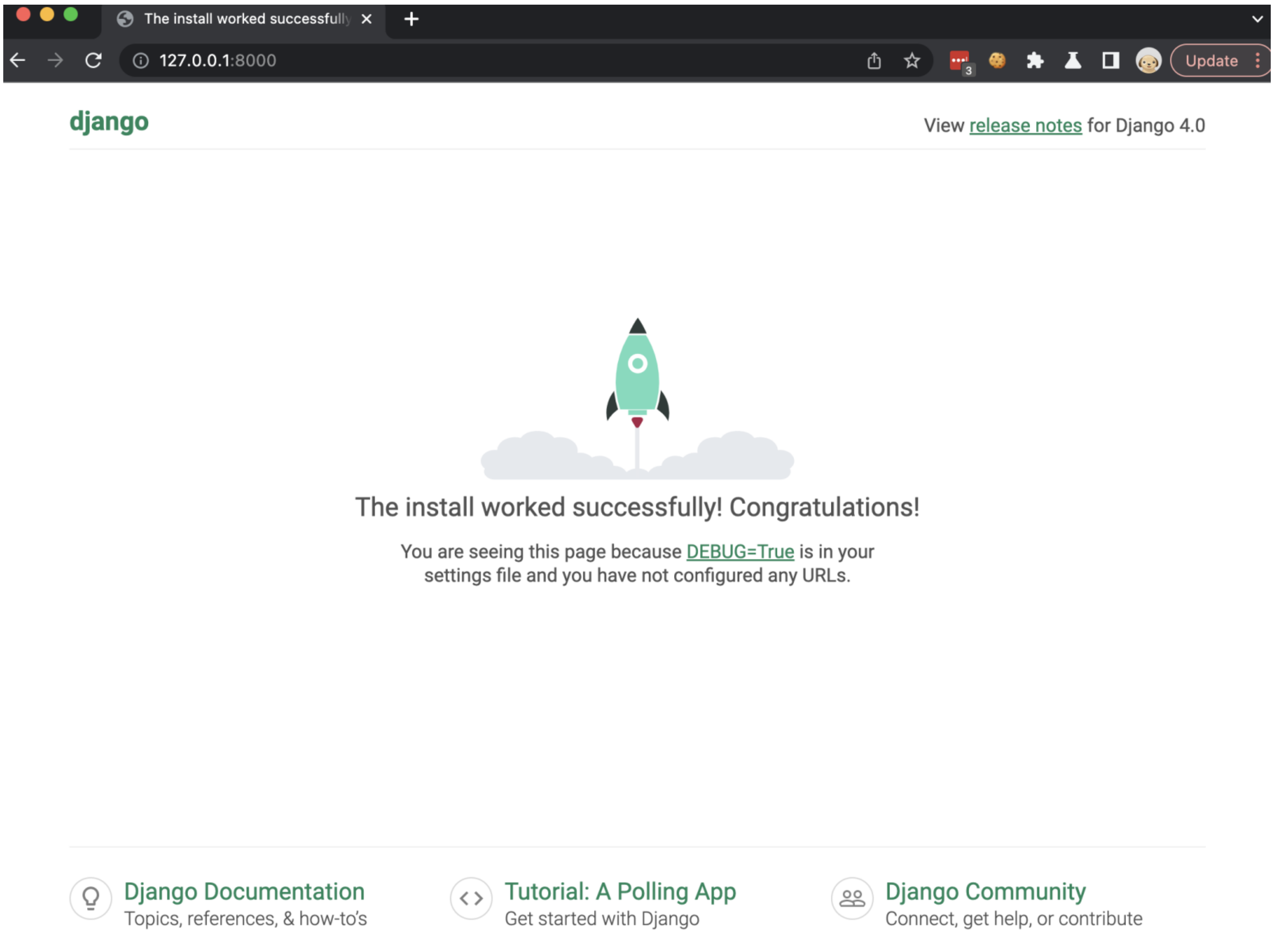
Even from this point we can start our project and it will work someway

```
$ 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.
```

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


The install worked successfully
127.0.0.1:8000
Update
django
View release notes for Django 4.0
Django Documentation
Topics, references, & how-to's
Tutorial: A Polling App
Get started with Django
Django Community
Connect, get help, or contribute

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.

Add IRIS

Let's add access to IRIS, and to do it we need to install a few dependencies to our project, and the right way to do it, is to define it in file named *requirements.txt with this content, where we should add django as a dependency*

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

And next, the driver for IRIS for Django, published. *Unfortunately, InterSystems does not want to publish own drivers to PyPI, so, we have to define it in this ugly way. Be aware, that they can delete it at any time, so, it may fail to work in future. (If it would be in pypi, it would be installed as a dependency of django-iris, and would not be required defined explicitly)*

```
# 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
```

Install dependencies defined in this file with command

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

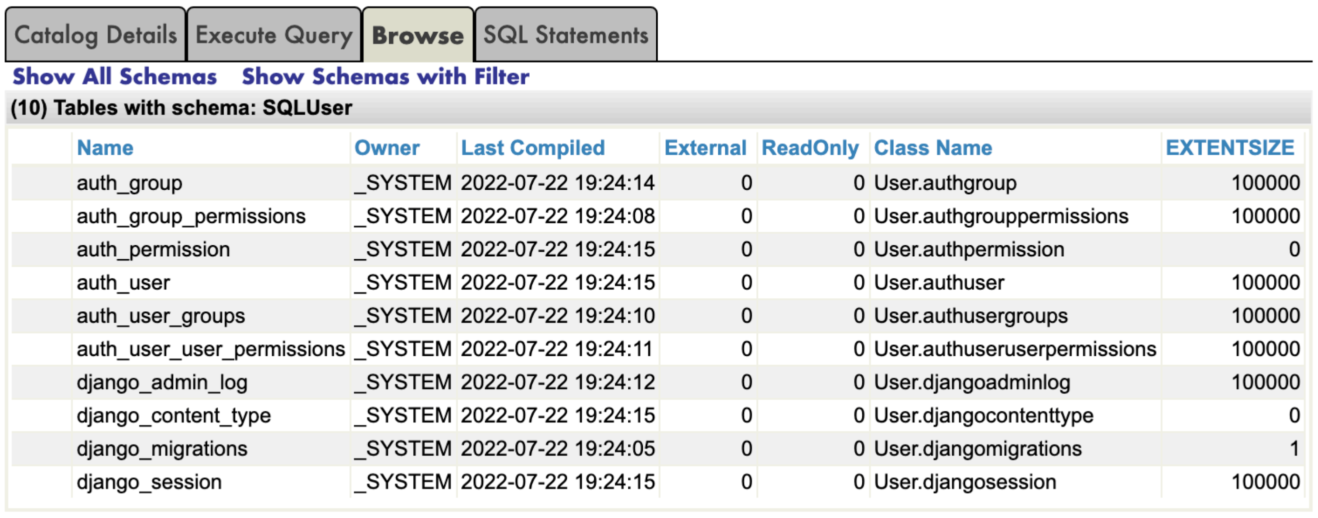
Now, we can configure our project to use IRIS, to do so, we have to update **DATABASES** parameter in *settings.py* file, with lines like this, where NAME is points to Namespace in IRIS, port to SuperPort where is IRIS available

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

Django has ORM, and models stored in the project, and it requires synchronizing Django models with Database as tables. By default, there are a few models, related to auth. And we can run migrate now

```
$ 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
```

If you would go to the IRIS, you would find some extra tables there


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

Define more models

It's time to add some of our models. To do so, add a new file *models.py,* with content like this

```
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()
```

As you can see, it has different types of fields. Then this model has to be prepared for Database. Before doing it, add our project **main** to **INSTALLEDAPPS** in *settings.py*

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

And we can makemigrations. This command has to be called after any changes in the model, it takes care of historical changes in the model, and no matter what the version of the application is installed, migration will know how to update the schema in the database

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

We can run migrate again, it already knows, that the previous migrations are already done, so, executes only new one

```
$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, main, sessions
Running migrations:
  Applying main.0001_initial... OK
```

And actually, we can see how migration looks from SQL view

```
$ 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);
```

But it is possible to get access to the tables already existing in the database, for instance, if you already have a working application. I have zpm package posts-and-tags installed, let's make a model for the table 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'
```

It is marked as **managed = False**, which means that makemigrations and migrate will not work for this table. Omitting the table name, will produce a big list of modules, including the tables already created by Django previously

#Python #InterSystems Ideas Portal #InterSystems IRIS
Check the related application on InterSystems Open Exchange

Source URL:https://community.intersystems.com/post/introduction-django-part-1