Introduction to Database Systems: CS312 Django: Setting Up An App Continued

Oliver BONHAM-CARTER

19th Oct. 2020

Steps to set up a virtual environment
MacOS and Linux Commands

Some commands may require `sudo` for *superuser*

- Install the virtual environment software. Done once. (See next slide.)
  - `pip install virtualenv`
  - or, `pip3 install virtualenv`
- Setup: Create an environment `myenv` for use with python3
  - `virtualenv myenv –p python3`
- Activate the environment
  - `source myenv/bin/activate`
- Install the Django software packages. Done once for each `env`.
  - `pip install django`
  - or, `pip3 install django`

More on Python virtual environments

Another way to set up a virtual environment (called, “myvenv”)

Some versions of Python already have *virtualizing* software already installed

- python -m venv myvenv
- or, python3 -m venv myvenv

Steps to set up a virtual environment
Windows Commands

- Install the virtual environment software. Done once. (See prev. slide.)

  - `pip install virtualenv`

- Setup: Create an environment `myenv` for use with python3

  - `virtualenv myenv –p python3`

- Activate the environment

  - `cd myenv/Scripts/`
  - Execute: `activate`

- Install the Django software packages. Done once for each env.

  - `pip install django`

Setting-up Django

Your terminal should now say, (myenv)

Install Django in your virtualenv

```
python -m django --version # check version
#or, python3 -m django --version # check version
```

Create your first Django project!

```
django-admin startproject mysite
```

Use manage.py to run the webserver to see your project!

```
cd mysite/
# we are now in: djangoWorking/myenv/mysite
python  manage.py runserver
# or, python3 manage.py runserver
```

Use your browser to check your work

```
http://127.0.0.1:8000/
Control-c to exit
```

Add Some Functionality

- We will now add an App to make the web site do something useful.
- We will add a Music App

Create The Music App

- Change into the *mysite* directory to locate the file `manage.py`, if you are not already there.

```
cd mysite/
python manage.py startapp music
```

The Files of Your App

```
find . -not -path '*/\.*'
```

```
----------------------------
./db.sqlite3
./manage.py
./music
./music/__init__.py
./music/admin.py
./music/apps.py
./music/migrations
./music/migrations/__init__.py
./music/models.py
./music/tests.py
./music/views.py
```

The Files of Your App

Notable Files

- apps.py: The main file for the music App
- models.py: A blueprint for how data will be used in the site
- tests.py: For adding tests for bug checking the music part of the project
- views.py: A request-handler for connecting the URL to the displayed website

mysite/mysite/urls.py and mysite/music/urls.py

- When the user enters a URL address, the website needs to know what pages to display.
- The *music/urls.py* file and the *mysite/mysite/urls.py* files are used to hold this URL-to-webpage connection information.
- We have to create the *mysite/music/urls.py* file for to connect the music urls to those of the entire website.

mysite/mysite/urls.py

```
from django.conf.urls import include, url
from django.contrib import admin
urlpatterns = [
# url(r'', admin.site.urls),
 url(r'^admin/', admin.site.urls),
 url(r'^music/', include('music.urls'))
]
```

- Note that the *music.urls* is an object, not a file.
- Be sure to use the correct quotation marks and add `include` to the top import statement.

mysite/music/urls.py

```
from django.conf.urls import include, url
# pull the local views.py file from local dir
from . import views
urlpatterns = [
 url(r'^$', views.index, name = 'index')
]
```

- The file, *music/urls.py* does not exist. You must create this file first.
- Be sure to use the correct quotation marks and add `include` to the top import statement.

mysite/music/views.py

```
from django.http import HttpResponse

def index(request):
my_str = "<h1> The Music App's homepage </h1>"
return HttpResponse(my_str)
```

- The file, *music/views.py* shows an html website
- Also, watch out that you are using the correct quotation marks! Some characters do not work with Python ...

Connect Your Databases

- When we start a project, there are *migration* problems.
- Meaning that the internal databases have not been connected to the web server.
- The database exists, but is not connected.
- To *make migrations* is to connect DB to the site to record events.

SQLite3 DB is defined in mysite/settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Seemingly, any database system could be used in Django by editing this code.

Create a Super User to View Database

Need to create database to remember user activity

- python manage.py migrate

Need to make an admin user for the site

- python manage.py createsuperuser

- Migrate is to connect a database to the project to hold user (admin) data
- Username (leave blank to use 'user'): admin
- Email address: studentID@allegheny.edu
- Password: *"pass1234"*
- Password (again): *"pass1234"*
- Superuser created successfully.
- Now, look around the admin page, `http://127.0.0.1:8000/admin`

A Working Website?

Use BASH command to see how DB was updated

- `sqlite3 db.sqlite3 "SELECT * FROM auth_user"`

- Restart the server: `python manage.py runserver`
- Enter the local URL in your browser: `http://127.0.0.1:8000`

Try these URLs

- `http://127.0.0.1:8000/music`
- `http://127.0.0.1:8000/admin`

Add Database Support

What to do with the database?

- Add to the Music App
  - Add Albums and Music classes (used to create SQLite3 tables)
  - Attributes:
    - Album: {Artist, AlbumTitle, genre, albumLogo }
    - Songs: {Album(foreignKey),fileType, songTitle}
- Use a browser to view and alter our data

Set up Models

Database's schema for data

- The *music/models.py* file contains variables in Python which become attributes in SQLite table.
- Django automatically creates the tables in SQLite.
- We set these attributes for the *music* table by editing the *music/models.py* file.

Set up Models

Database's schema for data

Watch out for proper indenting

music/models.py (part 1 of 2)

```
from django.db import models
class Album(models.Model):
   # holds the name of max length 250 chars
   artist = models.CharField(max_length = 250)
##
   # holds album name
   album_title = models.CharField(max_length = 500)
##
   # holds the genre
   genre = models.CharField(max_length = 100)
##
   # holds a url for music logo (link to graphic)
   album_logo = models.CharField(max_length = 1000)
#end of class Album()
```

Set up Models

Database's schema for data:

Watch out for proper indenting

music/models.py (part 2 of 2)

```
class Song(models.Model):
 # Class links the songs to the album class
 # foreign keys link the songs to a particular album.
 # when you delete an album, remove its
 # associated songs, as well.
##
   album = models.ForeignKey(Album,
   on_delete=models.CASCADE) # all on one line
##
   # holds the type of file containing music
   file_type = models.CharField(max_length = 10)
##
 # holds the song title.
   song_title = models.CharField(max_length = 250)
# end of class Song ()
```

mysite/settings.py

- The music app must work with a connected database.
- We add a line to the *INSTALLED_APPS* in *mysite/settings.py* to make this connection.

mysite/settings.py

```
INSTALLED_APPS = [
    # we have added this top line to the rest below
    'music.apps.MusicConfig', # Link Music App to DB
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Migrations Setup

- Error: *You have xx unapplied migration(s)...*
- After the changes, connection to databases must be build and made. (Use *makemigrations* for this)
- We need to install these tables. (Use *migrate* for this)

mysite/manage.py

```
python manage.py makemigrations music
python manage.py migrate
```

Output

```
Migrations for 'music':
  music/migrations/0001_initial.py:
    - Create model Album
    - Create model Song
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK ...
```

Migrations Setup

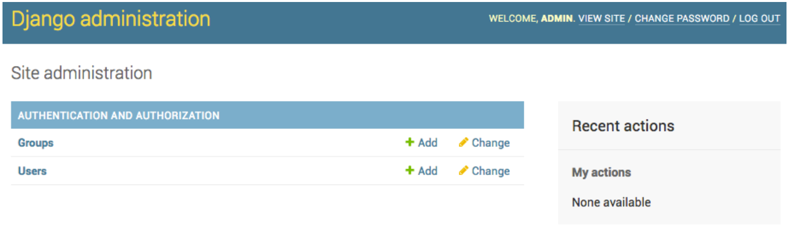
Show me the database schema!

```
python manage.py sqlmigrate music 0001
```

Partial output

```
CREATE TABLE "music_album" (
"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
 "artist" varchar(250) NOT NULL,
 "album_title" varchar(500) NOT NULL,
 "genre" varchar(100) NOT NULL,
 "album_logo" varchar(1000) NOT NULL);BEGIN;
--
--
CREATE TABLE "music_song" (
"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
"file_type" varchar(10) NOT NULL,
"song_title" varchar(250) NOT NULL,
"album_id" integer NOT NULL REFERENCES "music_album" ("id") DEFERRABLE INITIALLY DEFERRED);
CREATE INDEX "music_song_album_id_62a413c8" ON "music_song" ("album_id");
COMMIT;
```

Register your Database with the Project


Django administration
WELCOME, **ADMIN**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups	+ Add	Change
Users	+ Add	Change

Recent actions

My actions

None available

- *Album* should be registered as an *admin* site
- Edit *music/migrations/admin.py*

mysite/music/admin.py

```
from django.contrib import admin
from .models import Album
# from music/models.py class
##
admin.site.register(Album)
```

Plug in (Register) the Databases

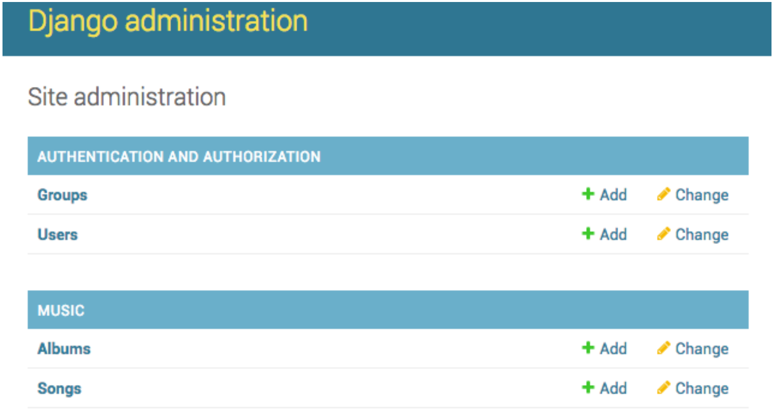
Just as before ...

- Now add the *Songs* database

File: `music/admin.py`

```
from django.contrib import admin
from .models import Album, Song
#
admin.site.register(Album)
admin.site.register(Song)
```

Check Your Web server By a Browser


Django administration

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups	+ Add	Change
Users	+ Add	Change

MUSIC

Albums	+ Add	Change
Songs	+ Add	Change

- `http://127.0.0.1:8000/admin`

Python Shell to Enter Data

- The database tables and schema are made by Django.
- We use Django's shell to check on the databases and the schemas.

python manage.py shell

```
--- then from Python ---
from music.models import Album, Song
Album.objects.all()
# gives   <QuerySet []> % empty; no data.
```

Adding Music Data Using Shell

- We are adding tuples into the database
- We enter the data using objects

Enter some data ...

```
from music.models import Album, Song
a = Album(artist = "The Nelsonions",
album_title = "Bluish-blue",
genre="Rock",
album_logo = "https://allegheny.edu/wp-content/uploads/1/2020/08/ic_logo-desktop@2x.png")
# above code is all on one line
##
a.save() # writes into the database
##
a.id # show the ID (primary key)
Album.objects.all() # there is something in now.
```

Adding More Music Data

- Add to the database one attribute at a time by a blank object.

Another way to enter data ...

```
b = Album()
b.artist = "The Beatles"
b.album_title = "The White Album"
b.genre = "rock"
b.album_logo = "https://upload.wikimedia.org/wikipedia/
   commons/thumb/d/df/The_Fabs.JPG/440px-The_Fabs.JPG"
#on one line
##
b.save()
```

Print Database Data in the Shell

Add this code to see query results in shell

- Printing *Album.objects.all()* acts like a "select" statement.
- Add code to print out album title and artist information with *Album.objects.all()*.

To see queries from shell, add to `music/models.py`

```
#The following is a method of Album class
# Place this code UNDER the Album class
   def __str__(self):
    # show the album_title, artist
    # return self.album_title+ ' +--+ ' + self.artist # shorter listing
      return self.artist + " +--+ " + self.album_title+ " +--+ " + self.artist
#end of __str__()
```

Usage from `manage.py shell`

```
>>>from music.models import Album, Song
>>>Album.objects.all()
<QuerySet [<Album: The White Album - The Beatles>]>
>>>Album.objects.filter(id=2)
<QuerySet [<Album: Bluish-blue - The Nelsonions>]>
```