

Notes from COSI Presentation on Django

April 30, 2011

1 Installing Django

With Python already installed (and for Ubuntu, apt-get install python-setuptools), run “sudo easy_install django”.

2 Creating a new project

Run “django-admin.py startproject < name of project >”. A new directory will be created within your current directory, which contains an empty project.

3 Changing the project settings

In “settings.py”:

3.1 Database

1. Change “ENGINE” to read “django.db.backends.sqlite3”
2. Change “NAME” to the full path of a database file, e.g. “/Users/chris/myproject/development.db”. The file will be created later.
3. Leave the remaining database fields blank

3.2 Media root

Set “MEDIA_ROOT” to the directory that will contain static media (CSS, images, Javascript, etc.), such as “/Users/chris/myproject/media/”. We will need this setting later for file uploads.

3.3 Template directories

Add the absolute path to the directory that will contain templates, e.g. “/Users/chris/myproject/templates/”. You will have to create this directory yourself.

3.4 Installed apps

For now, just uncomment “django.contrib.admin”.

4 Syncing the database

Run “python manage.py syncdb”, which creates the database file (if it doesn’t already exist), creates new tables and allows you to create an initial user (an administrator).

“syncdb” is used every time you create new models (or install new apps). One limitation is that it cannot *modify* models. If you add new fields or make changes to the actual structure of the data, you will have to drop the table by hand (with Navicat or similar) or delete the database file and resync. There are third-party pluggable apps¹ that can resolve this, but are somewhat complicated to use and so not covered here.

5 Setting up the URL mappings

In “urls.py”:

1. Uncomment lines 6 and 7 – This tells the built-in admin app to look through everything in the *INSTALLED_APPS* setting for models that are “registered” with the admin system (we’ll come back to this).
2. Uncomment the line that reads “url(r’^admin/’, include(admin.site.urls))”. This tells Django that any URL that starts with “admin/” is handled the “urls.py” inside the admin app module. Keeping the URL mapping for an app within its structure makes it pretty easy to keep apps entirely distinct within a project.

6 Running the development server

Run “python manage.py runserver” and then open <http://localhost:8000/admin/>, log in and explore.

Press Ctrl-C to exit. You can keep it running while you develop, however. It’s smart enough to watch the code for changes and show them immediately.

7 Creating the “images” app

7.1 Setting it up

1. Run “python manage.py startapp images”, where “images” is the app name. By convention these should not be capitalized, and if multiple words, separate with underscores. An “images” directory will be created within the project.
2. Add “images” to the end of *INSTALLED_APPS* in “settings.py”.

7.2 Creating the Model

In “models.py” within the “images” directory, add the contents of Figure 1

7.3 Validate the model

To ensure that the model checks out, run “python manage.py validate”, which checks for syntax, import and field errors. It’s generally wise to do this whenever models change, but is not strictly necessary.

7.4 Sync the database

Run “python manage.py syncdb”.

¹See <http://code.djangoproject.com/wiki/SchemaEvolution> for a comprehensive list and analysis

```

from django.db import models
from django.contrib.auth.models import User
from django.contrib import admin

# Create your models here.

class Image(models.Model):
    user = models.ForeignKey(User)
    posted = models.DateTimeField(auto_now_add=True)
    file = models.FileField(upload_to='media/upload')

    def __unicode__(self):
        return "%s posted by %s on %s" % (self.file.name, self.user, self.posted)
admin.site.register(Image)

```

Figure 1: Image Model

7.5 Creating the view

In “views.py” within the “images” directory, add the contents of Figure 2

```

from django.shortcuts import render_to_response

from models import Image

def index(request):
    # View body here
    images = Image.objects.all()
    return render_to_response('index.html', {'list_of_images': images})

```

Figure 2: Index View

The key here is that the array of images is called “images” within the view and “list_of_images” within the template. In the render_to_response function, the second parameter is defining what variables are passed to the template and what they will be named within the template.

7.6 Creating the template

In the “templates” directory that was created earlier, create a new file “index.html”, and enter the contents of Figure 3.

```

<html>
<head>
    <title>List of Images</title>
</head>
<body>
    <h1>Images</h1>
    <ul>
        {% for image in list_of_images %}
            <li>
                <a href="{{ image.file.url }}">{{ image.file.name }}</a>
                -- Posted by {{ image.user }} on {{ image.posted }}</li>
            {% endfor %}
        </ul>
    </body>
</html>

```

Figure 3: Index Template

7.7 Mapping “/images/” to the view

In “urls.py”, add the highlighted gray line from Figure 4 above the line that maps the admin app in place, and then add the remainder of the figure at the bottom of the file.

```
urlpatterns = patterns('',
    url(r'^images/$', 'images.views.index'),
    url(r'^admin/', include(admin.site.urls)),
)

import settings
if settings.DEBUG:
    urlpatterns += patterns('',
        url(r'^media/(?P<path>.*)$', 'django.views.static.serve', {
            'document_root': settings.MEDIA_ROOT,
        }),
    )
```

Figure 4: URLs

The last section tells Django to serve static media files (which is normally isn’t designed to do), but only while debug mode is on. In deployment, another web server (Apache, nginx or lighttpd) is used for efficiency, but it isn’t worth setting one of those up for development purposes.

8 Deployment

In deployment, your project is run by an application server (similar to Servlets), with a web server acting as the gateway. One of the better resources for information on deployment strategies and techniques is at <http://djangobook.com/en/2.0/chapter12/>.

It is wise to change *DEBUG* to False within “settings.py”, so that you don’t leak information when something breaks.

9 Third-party apps

There are many third-party apps out there that just “plug in” to a project:

1. django-registration² - Handles user registration, login, lost password, change password, etc.
2. DjangoBB³ - Forum app
3. Biblion⁴ - Blog app
4. And many others

10 Useful references

1. Official Django Documentation: <http://docs.djangoproject.com/en/1.3/>
2. The Django Book, 2nd Edition (Free web-edition): <http://djangobook.com/en/2.0/>
3. Any code-hosting site, search for “django-” and a whole bunch of open-source apps will show up
4. DjangoPackages.com: <http://djangopackages.com/> – Aggregates Django apps by keyword

²<https://bitbucket.org/ubernostrum/django-registration/>

³<http://djangobb.org/>

⁴<https://github.com/eldarion/biblion>