Django: Beyond the SQL

Jerry Stratton
February 26, 2011

Installation

Tutorial files

- RTF tutorial
- PDF tutorial
- Django tar.gz file
- Sample post titles, content, and topics
- Initial HTML template
- Smultron

For PHP programmers

Most of the common programming languages today resemble C. Braces are used to mark off blocks of code, such as for functions, loops, and conditional code. And semicolons are used to mark the end of a piece of code. That style of coding gives you a lot of flexibility in how you format your code, but over time standards have emerged; one of those standards is indentation, another is one one line per statement.

Django uses Python as its programming language. Python doesn't use braces or semicolons. Python was designed with the idea that since you're going to indent anyway, the braces are redundant. So a function that looks like this in PHP:

```
function englishJoin(myList) {
    lastItem = array_pop(myList);
    commaList = implode(', ', myList);
    if (count(myList) > 1) {
        theAnd = ', and ';
    } else {
        theAnd = ' and ';
    }
    englishList = commaList . theAnd . lastItem;
    return englishList;
}
```

might look like this in Python:

```
def englishJoin(myList):
    lastItem = myList.pop()
    commaList = ', '.join(myList)
    if len(myList) > 1:
        theAnd = ', and '
    else:
        theAnd = ' and '
    englishList = commaList + theAnd + lastItem;
    return englishList
```

Where PHP uses “function”, Python uses “def”. You’ll also notice lots of periods. Python uses classes a lot; just about everything is a class in Python. The period is the equivalent of PHP’s -> symbol for accessing properties and methods. So, .pop() is a method on lists, and .join() is a method on strings.

The latter one is a little weird: you don’t tell give a string a concatenator and tell it to pull itself together; you give a concatenator a string and tell it to get to work.

Classes in Python are similar to classes in PHP. Instead of “class ClassName { … }”, you use “class ClassName(object):”, which is the equivalent of PHP’s “class ClassName extends OtherClassName { … }”. In Python, all or almost all of the classes you create will extend another class, even if that class is just the basic “object”.

Requirements

You need Python 2.5 or 2.6, and SQLite 3. Any modern system should have these. If you have Mac OS X Leopard or Snow Leopard, for example, you have it.

You’ll need to perform the installation from your administrative account.

Install

http://www.djangoproject.com/download/

Download the latest Django from http://www.djangoproject.com/. As I write this, the latest release is 1.1.1. Download it; it will be Django-1.1.1.tar.gz. Use ‘gunzip’ to decompress it; then use tar -xvf to dearchive it. You can also just double-click it in Mac OS X to decompress and dearchive it.

Once you’ve got the Django-1.1.1 folder ready, you need to go to the command line. In Mac OS X, this is the Terminal app in your Utilities folder. Type ‘cd’, a space, and then the path to the Django folder (you can probably just drag and drop the folder onto the terminal window after typing ‘cd ’.

Once you’re in the Django folder in the command line, type “sudo python setup.py install”.

Django is now installed. If your administrative account is not your normal account, go back to your normal account.

Create a project

http://docs.djangoproject.com/en/dev/intro/tutorial01/

Use the command line to get to whatever folder you want to store your Django project in. Type “django-admin.py startproject Blog”. Django will create a “Blog” folder and put some initial files into it for you.

Change directory into the Blog folder, and start the development server:

```
python manage.py runserver
```

Watch the messages, and then go to a browser to the URL “http://localhost:8000/”. If it says “It worked!” then you’re good to go.

Edit settings

Your project has a file called “settings.py”. You need to change a few things there, mainly the name and location of the database you’re using.

Your DATABASE_ENGINE should be “sqlite3”. Your DATABASE_NAME should be “Blog.sqlite3”.

```
DATABASE_ENGINE = 'sqlite3'
DATABASE_NAME = 'Blog.sqlite3'
```

You’ll also need to tell Django what time zone you’re in:

```
TIME_ZONE = 'America/Los_Angeles'
```

Django comes with a wonderful administration tool for SQL databases, but you need to enable it. Head down to the bottom of settings.py and look for INSTALLED_APPS. Duplicate the last line inside the parentheses (it probably says “django.contrib.sites”) and change the new line to “django.contrib.admin”.

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.admin',
)
```

Go to the command line and type “python manage.py syncdb” inside your Blog project folder. This will create your database file. You’ll need to set up a superuser username, password, and e-mail. Remember your password!

If you look in your Blog project folder, you’ll see Blog.sqlite3; this is your SQLite database file.

Finally, open “urls.py” and remove the hash mark (comment) in front of the lines with the comment about enabling the admin:

```
from django.contrib import admin
admin.autodiscover()
...
(r'^admin/', include(admin.site.urls)),
```

Go to “http://localhost:8000/admin/” in your browser, and you should now see the basic administrative page for the built-in user databases.

Click here to download full PDF material