Practical Django Projects

ISBN-13 (pbk): 978-1-59059-996-9

ISBN-10 (pbk): 1-59059-996-9

ISBN-13 (electronic): 978-1-4302-0868-6

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editors: Steve Anglin, Tom Welsh
Technical Reviewer: Russell Keith-Magee
Editorial Board: Clay Andres, Steve Anglin, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Matthew Moodie, Joseph Ottinger, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager: Richard Dal Porto
Copy Editors: Kim Benbow, Nicole Abramowitz
Associate Production Director: Kari Brooks-Copony
Production Editor: Kelly Gunther
Compositor: Dina Quan
Proofreader: Nancy Sixsmith
Indexer: Carol Burbo
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales—eBook Licensing web page at `http://www.apress.com/info/bulksales`.

The source code for this book is available to readers at `http://www.apress.com`. You may need to answer questions pertaining to this book in order to successfully download the code.

CHAPTER 1

Welcome to Django

Web development is hard, and don't let anybody tell you otherwise. Building a fully functional dynamic web application with all the features users will want is a daunting task with a seemingly endless list of things you have to get just right. And before you can even start thinking about most of them, there's a huge amount of up-front work: you have to set up a database, create all the tables to store your data, plan out all the relationships and queries, come up with a solution for dynamically generating the HTML, work out how to map specific URLs to different bits of the code, and the list goes on. Just getting to the point where you can add features your users will see or care about is a vast and largely thankless job.

But it doesn't have to be that way.

This book will teach you how to use Django, a "web framework" that will significantly ease the pain of embarking on new development projects. You'll be able to follow along and build real applications—code you can actually use in the real world—and at every step you'll see how Django is there to help you out. And at the end, you'll come to a wonderful realization—that web development is fun again.

What's a Web Framework and Why Should I Want One?

The biggest downside of web development is the sheer amount of tedium it involves. All those things I've listed previously—database creation and querying, HTML generation, URL mapping—and dozens more are lurking behind every new application you develop, and they quickly suck all the joy out of even the most exciting projects. Web frameworks like Django aim to take all that tedium away by providing an organized, reusable set of common libraries and components that can do the heavy lifting, freeing you up to work on the things that make your project unique.

This idea of standardizing a set of common libraries to deal with common tasks is far from new. In fact, in most areas of programming it's such an established practice that you'd get strange looks if you suggested somebody should just start writing code from scratch. And in enterprise web development, frameworks of various sorts have been in use for years. Most companies that routinely need to develop large-scale applications rely heavily on frameworks to provide common functionality and speed up their development processes.

But in the world of web development, frameworks have traditionally been, almost out of necessity, just as heavyweight as the applications they're used in. They tend to be written in Java or C#, targeted at large corporate development projects, and sometimes come with a price tag that only a *Fortune* 500 company could love. Django is part of a new generation of

frameworks targeted at a broader audience: developers who don't necessarily have the weight of a multinational conglomerate's needs bearing down on their shoulders, but who still need to get things done quickly. Not to put too fine a point on it, developers like you and me.

The past couple of years have seen a number of these new web frameworks burst onto the scene, written in and for programming languages that are much more accessible to the average web developer (and, just as importantly, to the average web host): PHP, Perl, Python, and Ruby. Each one has a slightly different philosophy when it comes to things like code organization and how many "extras" should be bundled directly in the framework, but they all share a common baseline goal: provide an integrated, easy-to-use set of components that handle the tedious, repetitive tasks of web development with as little fuss as possible.

Say Hello to Django

Django began life as a simple set of tools used by the in-house web team of a newspaper company in a small college town in Kansas. Like anybody who spends enough time doing web development, they quickly got tired of writing the same kinds of code—database queries, templates, and the whole nine yards—over and over again, and extremely quickly, in fact, because they had the pressure of a newsroom schedule to keep up with. It wasn't (and still isn't) unusual to need custom code to go with a big story or feature, and the development timelines needed to be measurable in days, or even hours, in order to keep pace with the news.

In the space of a couple of years, they developed a set of libraries that worked extremely well together and, by automating or simplifying the common tasks of web development, helped them get their work done quickly and efficiently. In the summer of 2005, they got permission from the newspaper's management to release those libraries publicly, for free, and under an open source license so that anyone could use and improve them. They also gave it a snappy name, "Django," in honor of the famous gypsy jazz guitarist Django Reinhardt.

As befits its newsroom heritage, Django bills itself as "the web framework for perfectionists with deadlines." At its core is a set of solid, well-tested libraries covering all of the repetitive aspects of web development:

- An *object-relational mapper*, a library that knows what your database looks like, what your code looks like, and how to bridge the gap between them with as little hand-written SQL as possible.
- A set of HTTP libraries that knows how to parse incoming web requests and hand them to you in a standard, easy-to-use format and turns the results of your code into well-formed responses.
- A URL routing library that lets you define exactly the URLs you want and map them onto the appropriate parts of your code.
- A validation library for displaying forms in web pages and processing user-submitted data.
- A templating system that lets even nonprogrammers write HTML mixed with data generated by your code and just the right amount of presentational logic.

And that's just scratching the surface. Django's core libraries include a wealth of other features you'll come to love. A number of useful applications that build on Django's features are

also bundled with it and provide out-of-the-box solutions for specific needs like administrative interfaces and user authentication. In the example applications used in this book, you'll see all of these features, and more, in action. So let's dive in.

Say Hello to Python

Django is written in a programming language called Python, so the applications you develop with it will also be written in Python. That also means you'll need to have Python installed on your computer before you can get started with Django. Python can be downloaded for free from `http://python.org/download/` and is available for all major operating systems. It's best to install the latest version of Python—Python 2.5.1 at the time of this writing—in order to have the latest features and bug fixes for the Python language.

ADMONITION: LEARNING PYTHON

If you don't know any Python, or even if you've never done any programming before, don't worry. Python is easy to learn (when I first started with Python, I learned the basics in a weekend by reading online tutorials), and you don't need to know much of it to get started with Django. In fact, many first-time Django users learn Python and Django at the same time.

Throughout this book, I'll call attention to important Python concepts when needed, but it would be a good idea to look at a Python tutorial before going very far into this book. The Python documentation index (available online at `http://python.org/doc/`) has a good list of tutorials and books (several of which are available for free online) to help you learn the basics of Python. (I'd recommend knowing at least how Python functions and classes work.) You'll be able to pick up the rest as you go along.

If you're looking for a good reference to keep handy as you're learning Django, *Beginning Python: From Novice to Professional* by Magnus Lie Hetland, and *Dive Into Python* by Mark Pilgrim (both from Apress) are good options.

Once you've installed Python, you should be able to open a command prompt (Command Prompt on Windows, Terminal on Mac OS X, or any terminal emulator on Linux) and enter the Python interactive interpreter by typing the command **`python`**. Normally, you'll be saving your Python code into files to be run as part of your applications, but the interactive interpreter will let you explore Python—and, once it's installed, Django—in a more freeform way: the interpreter lets you type in Python code, a line at a time, and see the results immediately. You can also use it to access and interact with code in your own Python files or in the Python standard libraries and any third-party libraries you've installed, which makes it a powerful learning and debugging tool.

When you first fire up the Python interpreter, you'll see something like this:

```
Python 2.5.1 (r251:54869, Apr 18 2007, 22:08:04)
[GCC 4.0.1 (Apple Computer, Inc. build 5367)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The >>> is Python's command prompt. You can type a line of Python code and press Enter, and if that code returns a result, you'll see it immediately. Let's test this with a simple line that just prints a line of text. At the Python interpreter prompt, type the following and press Enter:

```
>>> print "Hello, world!"
```

You'll see the result appear on the next line:

```
Hello, world!
>>>
```

Anything you can type into a file as part of a Python program can be typed directly into the interpreter, and there's also a full help system built in, which you can access at any time by typing **`help()`** and pressing Enter. When you're ready to exit the Python interpreter, press Ctrl+D, and it will shut down.

Installing Django

Now that you've got Python installed and working, it's time to install Django and start exploring its features. You can get a copy from the official Django web site; just visit `www.djangoproject.com/download/` and follow the instructions for downloading the "development version" of Django.

ADMONITION: PACKAGED RELEASES VS. DEVELOPMENT CODE

Django is always being worked on and improved and, in addition to the official release, the current in-development code is available for download. The Django web site has instructions for installing this code on your computer, and you can follow that to obtain the development version of Django.

The advantage of using the development version is that new features are available as soon as they're added, so you can begin using them immediately instead of waiting for the next official release. In this book, I'll be assuming that you've installed the development version of Django, and several of the features we'll use are only available in the development version. This code will, in the near future, become Django's packaged 1.0 release, so starting out with it will minimize the amount of work you'll need to do to upgrade when that takes place. So when you download Django, be sure to follow the specific instructions for the development version found at www.djangoproject.com/documentation/install/#installing-the-development-version.

Once you've downloaded the Django code onto your computer, you can install it by typing a single command. On Linux or Mac OS X, open a terminal, navigate to the directory Django downloaded into, and you should see a file named `setup.py`. Type the following command, and enter your password when prompted:

```
sudo python setup.py install
```

On Windows, you'll need to open a command prompt with administrative privileges; then you can navigate to the Django directory and type the following:

```
python setup.py install
```

The `setup.py` script is a standard installation procedure for Python modules, and takes care of installing all of the relevant Django code into the correct locations for your operating system. If you're curious, Table 1-1 summarizes where the Django code will end up on various systems.

Table 1-1. *Django Installation Locations*

Operating system	Django location
Linux	`/usr/local/lib/python2.5/site-packages/django`
Mac OS X	`/Library/Frameworks/Python.framework/Versions/2.5/lib/python2.5/site-packages/django`
Windows	`C:\Python\site-packages\django`

Your First Steps with Django

You should now be able to verify that Django installed correctly on your computer. Next, start the interactive Python interpreter and type in the following:

```
>>> import django
>>> print django.VERSION
```

The result of this should be a set of numbers in parentheses, which represents the version of Django you're using. The Django 0.96 release, for example, will show (0, 96). Python software typically uses a *tuple*—a parenthesized, comma-separated list of numbers and/or words—to represent version numbers internally (which makes it easy for Python programs to automatically parse otherwise complex version numbers like "1.0 beta 3" or "2.4 prerelease").

Now you're ready to create your first Django project. A Django *project* is a wrapper of sorts, which contains settings for one or more Django-powered applications and a list of which applications it uses. Later on, when you're deploying your Django applications behind a real web server, you'll use projects to organize and configure them.

To set up your first project, create a directory on your computer where you'll keep your in-progress Django projects, and then navigate to it in a terminal or at a command prompt. It's often a good idea to have a single directory where you keep all of your own custom Python code. As you'll see a bit later on, doing so will simplify the process of telling Python how to find and use that code.

Now you can use the built-in Django management script, `django-admin.py`, to create your project. `django-admin.py` lives in the `bin/` subdirectory of the directory Django was installed into, and it knows how to handle various management tasks involving Django projects. The one you're interested in is called `startproject`, and it will create a new, empty Django project. In the directory where you want to create your project, type the following (refer to Table 1-1 for the correct path for your operating system):

```
/usr/local/lib/python2.5/site-packages/django/bin/django-admin.py startproject cms
```

This will create a new subdirectory called `cms` (you'll see why it's named that in the next chapter, when you start to work with this project) and populate it with the basic files needed by any Django project.

ADMONITION: PERMISSION ERRORS

If you're using Linux or Mac OS X, you may see an error message saying "permission denied." If this happens, you need to tell your operating system that the `django-admin.py` script is safe to run as a program. You can do this by navigating to the directory that `django-admin.py` is in and typing the command **`chmod +x django-admin.py`**. Then you can run the `django-admin.py` script as previously shown.

In the next section you'll see what each of the files in the project directory is for, but for now the most important one is called `manage.py`. Like `django-admin.py`, it's there to take care of common project and application management tasks for you. The `manage.py` script can start a simple web server that will host your project for testing purposes, and you can start it by going into your project directory and typing the following:

```
python manage.py runserver
```

Then you should be able to open up a web browser and visit the address `http://127.0.0.1:8000/`. The development web server, by default, runs on your computer's local "loopback" network address, which is always 127.0.0.1 and binds to port 8000. When you visit that address, you should see a simple page saying "It worked!" with some basic instructions for customizing your project (see Figure 1-1).

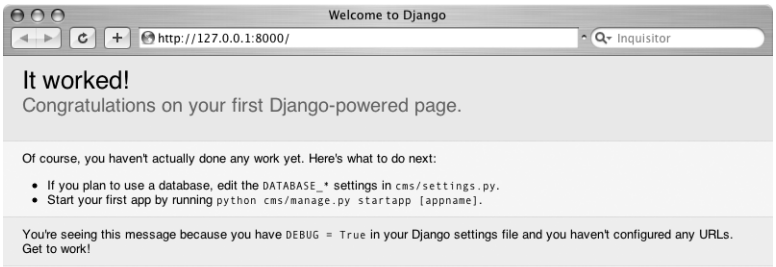

Figure 1-1. *Django welcome screen*

ADMONITION: CHANGING THE ADDRESS AND PORT

If something else is already using port 8000 on your computer, if you're not allowed to run programs that bind to that port, or if you want to view pages served by Django's development server from another computer, you'll need to manually specify the address and port to use when you launch the development server. The syntax for this is `python manage.py runserver ip_address:port_number`. So, for example, to listen on all of your computer's available IP addresses (so that other computers can view pages from the development server) and bind to port 9000 instead of 8000, you could type `python manage.py runserver 0.0.0.0:9000`.

You can stop the server by pressing Ctrl+C at the command prompt.

Exploring Your Django Project

The `startproject` command of `django-admin.py` created your project directory for you and automatically filled in a few files. Each one serves a specific purpose, and in future chapters you'll see what each one does, but for now here's a quick primer:

`__init__.py`: This will be an empty file. For now you don't need to put anything into it (and in fact, most of the time you won't need to). It's used to tell Python that this directory contains executable code. Python can treat any directory containing an `__init__.py` file as a Python module.

`manage.py`: As explained previously, this is a helper script that knows how to handle common management tasks. It knows how to start the built-in development web server, create new application modules, set up your database, and numerous other things that you'll see as you build your first Django applications.

`settings.py`: This is a Django *settings module*, which holds the configuration for your Django project. Over the next few chapters, you'll see some of the most common settings and how to edit them to suit your projects.

`urls.py`: This file contains your project's master URL configuration. Unlike some languages and frameworks that simply mimic HTML by letting you place code into the web server's public directory and access it directly by file name, Django uses an explicit configuration file to lay out which URLs point to which parts of your code, and this file defines the set of "root" URLs for an entire project.

You may notice that, after you started the built-in web server, one or more new files appeared in the project directory with the same names as those in the preceding list but with a .pyc extension instead of .py. Python can read the code directly out of your .py files, but it also can, and often does, automatically compile code into a form that's faster to load when a program starts up. This *bytecode*, as it's called, is then stored in identically named .pyc files, and if the original file hasn't changed since the last time a program used it, Python will load from the bytecode file to gain a speed boost.

Looking Ahead

In the next chapter, you'll walk through setting up your first real Django project, which will provide a simple content management system, or CMS. If you're ready to dive in, keep reading, but you should also feel free to pause and explore Python or Django a bit more on your own. Both the `django-admin.py` and `manage.py` scripts accept a `help` command, which will list all of the things they can do; and the Python interpreter's built-in help system can also automatically extract documentation from most Python modules on your computer, including the ones inside Django. There's also a special `shell` command to `manage.py` that you may find useful because it will launch a Python interpreter with a fully configured Django environment (based on your project's settings module) you can explore.

If you'd like, you can also take this opportunity to set up a database to use with Django. If you installed Python 2.5 or any later version, you won't have to do this right away. As of version 2.5, Python includes the lightweight SQLite database system directly, which you'll be able to use that throughout this book as you develop your first applications. However, Django also supports MySQL, PostgreSQL, and Oracle databases, so if you'd prefer to work with one of those, go ahead and set it up.