Practical Django Projects

James Bennett

Apress®

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.

This book would not have been possible without the huge and supportive community that has grown up around Django in the past three years. The willingness of people all around the world to freely contribute their code, their ideas, and their time to improving the state of our art never ceases to amaze me.

This book also would not have been possible without Mr. Morgan, who instilled in me both the craft and the joy of writing. For that he has my deepest thanks.

Contents at a Glance

Contents

About the Author

JAMES BENNETT is a web developer at the *Lawrence Journal-World* in Lawrence, Kansas, where Django was originally developed. He is both a regular contributor to and the release manager for the open source Django project.

About the Technical Reviewer

DR. RUSSELL KEITH-MAGEE has been a core developer on the Django project since January 2006. He is a cofounder of Django Evolution, a schema evolution framework for Django. He is an active participant on the Django Users and Django Developers mailing lists and is a mentor in the Google Summer of Code 2008.

In addition to his work with Django, Russell has worked at two startup companies—one very successful and one still in development. In those jobs, he has used his passion for good design, powerful tools, and automated testing to find elegant solutions to real-world problems faced by real-world users.

Russell lives with his wife, son, and two cats in Perth, Western Australia.

Introduction

The past few years have seen an explosion in the development of dynamic, database-driven web sites. Where many sites were once built using nothing but handwritten HTML, or a few CGI scripts or server-side includes, today database-backed web applications have become the norm for everything from personal blogs to online stores to the social networking sites that have revolutionized the way many people use the Web.

But this has come at a cost. Developing these applications, even for relatively simple uses, involves a significant amount of complex work, and much of that work ends up being repeated for each new application. Although web developers have always had access to libraries of code that could automate certain tasks, such as HTML templating or database querying, the process of bringing together all the necessary pieces for a fully polished application has largely remained difficult and tedious.

This has led to the recent development, and subsequent popularity, of "web frameworks," reusable collections of components that handle many of the common and repetitive tasks of application development in an integrated fashion. Instead of requiring you to obtain disparate libraries of code and find ways to make them work together, web frameworks provide all the necessary components in a single package and take care of the integration work for you.

Django is one of the most recent crop of web frameworks, growing out of the needs of a fast-paced online news operation. Django's original developers needed a set of tools that would not only help them quickly develop new and highly dynamic web applications in response to the rapidly evolving requirements of the news industry, but would also let them save time and effort by reusing pieces of code, and even entire applications, whenever possible.

In this book, you'll see how Django can help you achieve both of these goals—rapid application development and flexible, reusable code—through the tools it provides to you directly and the development practices that it makes possible. I'll guide you through the development of several example applications and show you how the various components and applications bundled with Django can help you to write less code at each stage of the development process. You'll also see firsthand a number of best practices for reusable code and learn how you can apply them in your own applications, as well as see how to integrate existing third-party libraries into Django-powered applications to minimize the amount of code you'll need to write from scratch.

I've written this book from a pragmatic viewpoint. The sample applications are all intended to be useful in real-world situations, and once you've worked through them, you'll have more than just a technical understanding of Django and its components. You'll have a clear understanding of how Django can help you become a more productive and more effective developer.

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).

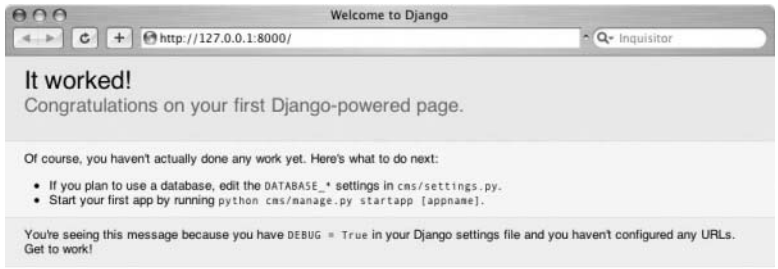

Figure 1-1. *Django welcome screen*

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.

CHAPTER 2

Your First Django Site: A Simple CMS

One extremely common task in web development is building a simple content management system (CMS), which lets users create and edit pages on a site dynamically through a web-based interface. Sometimes called *brochureware* sites because they tend to be used in the same fashion as traditional printed brochures handed out by businesses, they're usually fairly simple feature-wise, but can be tedious to code over and over again.

In this chapter, you'll see how Django makes these sorts of sites almost trivially easy to build: I'll walk you through the setup of a simple CMS, and then in the next chapter, you'll see how to add a few extra features and provide room to expand it in the future.

Configuring Your First Django Project

In the last chapter, you created a Django project called `cms`. But before you can do much with it you'll need to do some basic configuration, so launch your favorite code editing program and use it to open up the `settings.py` file in your project.

ADMONITION: WRITING PYTHON

From here to the end of this book, you'll be writing Python code and the occasional template. If you haven't already looked at a Python tutorial to get a feel for the basics, now would be a good time. I'll explain some of the most important concepts as we go, but that's no substitute for a dedicated Python tutorial, which will cover them in depth.

And if you don't have an editing program suitable for working with programming code, you'll want to get one. Nearly all programmers' editors have support for Python (and other popular languages) built in, and this will make the process of writing code much easier.

Don't be daunted by the size of this file or the number of things you'll find in it. `django-admin.py` automatically filled in default values for a lot of them, and for now most of the defaults will be fine. Near the top of the file is a group of settings whose names all start with `DATABASE`. These settings tell Django what type of database to use and how to connect to it, and right now that's all you'll need to fill in.

Assuming you installed the latest version of Python, you'll already have a database adapter module that can talk to SQLite databases (Python 2.5 and later include this module in the standard Python library). SQLite stores the entire database in a single file on your computer and doesn't require any of the complex server or permissions setup of other database systems, so it's a great system to use when you're just starting out or exploring Django.

To use SQLite, you'll only need to change two settings. First, find the `DATABASE_ENGINE` setting and change it from this:

```
DATABASE_ENGINE = ''
```

to this:

```
DATABASE_ENGINE = 'sqlite3'
```

Now you'll need to tell Django where to find the SQLite database file. This goes into the `DATABASE_NAME` setting and can be anywhere on your computer's hard drive where you have permission to read and write files. You can even fill in a nonexistent file name, and the SQLite database engine will create the file for you automatically. Keeping the database file inside your project folder isn't a bad idea in this case, so go ahead and do that. I keep all of my Django projects in a folder called `django-projects` inside my home directory (on a laptop running Mac OS X), so I'll fill it in like so:

```
DATABASE_NAME = '/Users/jbennett/django-projects/cms/cms.db'
```

On other operating systems this will look a bit different, of course. On Windows it might be `C:\Documents and Settings\jbennett\django-projects\cms\cms.db`, for example, while on a Linux system it might be `/home/jbennett/django-projects/cms/cmd.db`.

I'm telling Django the SQLite database file should live inside the cms project directory and be named `cms.db`. The .db file extension isn't required, but it helps me to remember what that file is for, and so I'd recommend you use something similar.

ADMONITION: USING A DIFFERENT DATABASE

If you'd like to set up a MySQL, PostgreSQL, or Oracle database instead of using SQLite, consult the Django settings documentation online at `www.djangoproject.com/documentation/settings/` to see the correct values for the database settings. However, bear in mind that you will also need to install a Python adapter module for the database you're using—as of Python 2.5, SQLite is the only database system directly supported in the standard Python library.

If you're using a version of Python prior to 2.5, you'll need to install an adapter module for your database no matter which database you use. See the Django installation instructions for details at `www.djangoproject.com/documentation/install/#get-your-database-running`.

Finally, you'll probably want to change the `TIME_ZONE` setting. This tells Django which time zone to use when displaying dates and times from your database (which are typically stored in your database as UTC timestamps—Universal Time, Coordinated, which is the "base" time zone formerly known as Greenwich Mean Time, or GMT). Rather than using a country-specific time-zone name (like US Central Time) or a confusing UTC offset (like UTC-6), this setting

uses names in zoneinfo format; *zoneinfo* is a standard format used by many computer operating systems and is also easy for humans to read. The default setting is

```
TIME_ZONE = "America/Chicago"
```

which is equivalent to the US Central time zone, six hours behind UTC. Full lists of zoneinfo time zone names are available online, and the official Django settings documentation at `www.djangoproject.com/documentation/settings/` includes a link to one such list. You should change your `TIME_ZONE` setting to the zone in which you live.

ADMONITION: TIME ZONES ON WINDOWS

If you're using Microsoft Windows, you'll want to be careful with the `TIME_ZONE` setting. Because of quirks in Windows' operating environment, it's not possible to reliably use a time zone other than the one the computer as a whole is currently using. So for best results you'll want to specify `TIME_ZONE` to be the same as the time zone Windows is using.

You won't need to change it yet, but you'll also want to scroll down to the bottom of the settings file, where you'll see a setting called `INSTALLED_APPS`. As mentioned previously, a Django project is made up of one or more Django-powered applications, and this setting is how Django knows which applications are used by your project. The default value looks like this:

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

Each of these is an application bundled with Django itself, and each one provides a useful piece of common functionality; `django.contrib.auth`, for example, provides a mechanism for storing data about users and for authenticating them, while `django.contrib.sites` provides an easy way to run multiple web sites from a single Django project and to specify which items in your database should be accessible to each site.

In time, you'll see examples of these applications in action, but for now it's best to leave the defaults as they are. They provide a "quick start" to your project by taking care of a lot of tasks right away, and you'll be building on their functionality in just a moment.

Now that you've provided some basic configuration data to Django, you can tell it to set up your database. Open up a terminal or command prompt, navigate to your project's directory, and type this command:

```
python manage.py syncdb
```

This command will create the database file if needed and then create the database tables for each application listed in the `INSTALLED_APPS` setting. You'll see several lines of output scroll by, and then, because the bundled user authentication application is being installed,

Django will ask if you'd like to create a "superuser" account for web-based administration. Type **yes**, and then enter a username, e-mail address, and password when prompted. You'll see shortly how you can use this account to log in to a Django administrative interface.

ADMONITION: WHAT GOES ON DURING SYNCDB

When you run `manage.py syncdb`, Django actually does several things in order, and the output on your screen shows each step. First, Django looks in each application module listed in `INSTALLED_APPS` and finds the *data models*. These are Python classes that define the different types of data the application uses, and Django knows how to automatically generate appropriate `CREATE TABLE` SQL statements from them. In Chapter 3, you'll write your first data model and see how Django generates the SQL for it.

Once the database tables have been created, Django looks for, and runs, any application-specific initialization code for each application. In this case, `django.contrib.auth` includes code that prompts you to create a user account.

Finally, Django finishes the database setup and installs any initial data you've provided. The default set of bundled applications doesn't use this feature, but later on you'll see how to supply an initial data file that can kick-start an application by giving it data to work with right away. You won't be providing any initial data with this application, but some of Django's bundled applications do provide data which will be inserted into the database when installed.

Putting Together the CMS

Most of the applications you'll build with Django will require you to write a fair amount of code on your own. Django will take care of the heavy lifting and the repetitive tasks, but it'll still be up to you to handle features unique to each specific application. Sometimes, though, features built in to Django or applications bundled with it will provide most or all of what you need. Django's `contrib` applications are designed with just this aim in mind: some types of applications are so common and so repetitive that it's best to just provide a single customizable version and reuse it from project to project.

A simple brochureware CMS is a good example of this, and you'll build it by relying heavily on two applications bundled with Django: `django.contrib.flatpages` and `django.contrib.admin`.

The first of these, `django.contrib.flatpages`, provides a data model for a simple page, with a title, content, and a few configurable options, such as custom templates or authentication. The other, `django.contrib.admin`, provides a powerful administrative interface that can work with any Django data model, letting you create a more or less "instant" web-based interface to administer a site.

The first step here is to add these applications to the `INSTALLED_APPS` setting. You'll remember that by default four applications were placed in the list, and now you can add two more:

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

```
    'django.contrib.sites',
    'django.contrib.admin',
    'django.contrib.flatpages',
)
```

Once you've made that change and saved your settings file, run `syncdb` again:

```
python manage.py syncdb
```

You'll see the output scroll by as Django creates database tables for the data models defined in these applications. Now, open up the `urls.py` file in your project, which—as you saw in the previous chapter—contains the root URL configuration for your project. There's a line that says, "Uncomment this for admin:" followed by this line (the hash mark at the beginning indicates a Python comment and means the line will not be executed as code):

```
#     (r'^admin/', include('django.contrib.admin.urls')),
```

Uncomment that line and save the file. This will add a set of URLs, included in `django.contrib.admin`, to your project's URL configuration.

ADMONITION: HOW DJANGO URL CONFIGURATION WORKS

A Django URL configuration file, or `URLConf`, defines a list of URL patterns and indicates how they map to parts of your code. Each URL pattern has at least two parts: a regular expression that describes what the URL looks like and either a view (a Python function that can respond to HTTP requests) to map that URL to or an `include`, which points to a different `URLConf` module. The ability to include other `URLConf` modules makes it easy to define reusable and "pluggable" sets of URLs, which can be dropped into any point in your project's URL hierarchy.

A *regular expression*, in case you've never encountered that term before, is a common way to represent a particular pattern of text, and most programming languages have support for checking whether a given piece of text matches the pattern specified in a regular expression. Most introductory programming books cover regular expressions. *Dive Into Python* by Mark Pilgrim (Apress, 2004) has a good chapter that covers the basics.

Also, note that regular expressions are quite strict about matching. Ordinarily, a web server will be somewhat lax and treat, for example, `/admin` and `/admin/` as the same URL, returning the same result either way. But if you specify a regular expression that ends in a slash—as I'm doing here—you *must* include the slash on the end when you visit that address in your browser, or the pattern will not match and you'll get a "Page not found" error.

Now you'll be able to launch the built-in web server again and see the administrative interface:

```
python manage.py runserver
```

The URL pattern for the admin application is `^admin/`, which means that if you visit `http://127.0.0.1:8000/admin/` in your web browser, you'll see the login page. Enter the username and password you used when `syncdb` prompted you to create a user account, and you'll