Web framework for Python

Django Book: pdf version

compiled by Suvash Sedhain

bir2su.blogspot.com

Visit www.djangobook.com for online version of the book

The Django Book

Table of contents

Beta, English

The Django Book

Chapter 1: Introduction to Django

If you go to the Web site djangoproject.com using your Web browser — or, depending on the decade in which you're reading this destined-to-be-timeless literary work, using your cell phone, electronic notebook, shoe, or any Internet-superceding contraption — you'll find this explanation:

> "Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design."

That's a mouthful — or eyeful or pixelful, depending on whether this book is being recited, read on paper or projected to you on a Jumbotron, respectively.

Let's break it down.

Django is a high-level Python Web framework…

A high-level Web framework is software that eases the pain of building dynamic Web sites. It abstracts common problems of Web development and provides shortcuts for frequent programming tasks.

For clarity, a dynamic Web site is one in which pages aren't simply HTML documents sitting on a server's filesystem somewhere. In a dynamic Web site, rather, each page is generated by a computer program — a so-called "Web application" — that you, the Web developer, create. A Web application may, for instance, retrieve records from a database or take some action based on user input.

A good Web framework addresses these common concerns:

- **It provides a method of mapping requested URLs to code that handles requests.** In other words, it gives you a way of designating which code should execute for which URL. For instance, you could tell the framework, "For URLs that look like `/users/joe/`, execute code that displays the profile for the user with that username."
- **It makes it easy to display, validate and redisplay HTML forms.** HTML forms are the primary way of getting input data from Web users, so a Web framework had better make it easy to display them and handle the tedious code of form display and redisplay (with errors highlighted).
- **It converts user-submitted input into data structures that can be manipulated conveniently.** For example, the framework could convert HTML form submissions into native data types of the programming language you're using.
- **It helps separate content from presentation via a template system**, so you can change your site's look-and-feel without affecting your content, and vice-versa.
- **It conveniently integrates with storage layers** — such as databases — but doesn't strictly require the use of a database.
- **It lets you work more productively, at a higher level of abstraction**, than if you were coding against, say, HTTP. But it doesn't restrict you from going "down" one level of abstraction when needed.
- **It gets out of your way**, neglecting to leave dirty stains on your application such as URLs that contain ".aspx" or ".php".

Django does all of these things well — and introduces a number of features that raise the bar for what a Web framework should do.

The framework is written in Python, a beautiful, concise, powerful, high-level programming language. To develop a site using Django, you write Python code that uses the Django libraries. Although this book doesn't include a full Python tutorial, it highlights Python features and functionality where appropriate, particularly when code doesn't immediately make sense.

…that encourages rapid development…

Regardless of how many powerful features it has, a Web framework is worthless if it doesn't save you time. Django's philosophy is to do all it can to facilitate hyper-fast development. With Django, you build Web sites in a matter of hours, not days; weeks, not years.

This is possible largely thanks to Python itself. Oh, Python, how we love thee, let us count the bullet points:

- Python is an **interpreted language**, which means there's no need to compile code. Just write your program and execute it. In Web development, this means you can develop code and immediately see results by hitting "reload" in your Web browser.
- Python is **dynamically typed**, which means you don't have to worry about declaring data types for your variables.
- Python syntax is **concise yet expressive**, which means it takes less code to accomplish the same task than in other, more verbose, languages such as Java. One line of python usually equals 10 lines of Java. (This has a convenient side benefit: Fewer lines of code means fewer bugs.)
- Python offers **powerful introspection and meta-programming** features, which make it possible to inspect and add

behavior to objects at runtime.

Beyond the productivity advantages inherent in Python, Django itself makes every effort to encourage rapid development. Every part of the framework was designed with productivity in mind. We'll see examples throughout this book.

..and clean, pragmatic design

Finally, Django strictly maintains a clean design throughout its own code and makes it easy to follow best Web-development practices in the applications you create.

That means, if you think of Django as a car, it would be an elegant sports car, capable not only of high speeds and sharp turns, but delivering excellent mileage and clean emissions.

The philosophy here is: Django makes it easy to do things the "right" way.

Specifically, Django encourages loose coupling: the programming philosophy that different pieces of the application should be interchangeable and should communicate with each other via clear, concise APIs.

For example, the template system knows nothing about the database-access system, which knows nothing about the HTTP request/response layer, which knows nothing about caching. Each one of these layers is distinct and loosely coupled to the rest. In practice, this means you can mix and match the layers if need be.

Django follows the "model-view-controller" (MVC) architecture. Simply put, this is a way of developing software so that the code for defining and accessing data (the model) is separate from the business logic (the controller), which in turn is separate from the user interface (the view).

MVC is best explained by an example of what *not* to do. For instance, look at the following PHP code, which retrieves a list of people from a MySQL database and outputs the list in a simple HTML page. (Yes, we realize it's possible for disciplined programmers to write clean PHP code; we're simply using PHP to illustrate a point.):

```
<html>
<head><title>Friends of mine</title></head>
<body>

<h1>Friends of mine</h1>

<ul>

<?php
$connection = @mysql_connect("localhost", "my_username", "my_pass");
mysql_select_db("my_database");
$people = mysql_query("SELECT name, age FROM friends");
while ( $person = mysql_fetch_array($people, MYSQL_ASSOC) ) {
?>
<li>
<?php echo $person['name'] ?> is <?php echo $person['age'] ?> years old.
</li>
<?php } ?>

</ul>

</body>
</html>
```

While this code is conceptually simple for beginners — because everything is in a single file — it's bad practice for several reasons:

1. **The presentation is tied to the code.** If a designer wanted to edit the HTML of this page, he or she would have to edit this code, because the HTML and PHP core are intertwined.

   By contrast, the Django/MVC approach encourages separation of code and presentation, so that presentation is governed by templates and business logic lives in Python modules. Programmers deal with code, and designers deal with HTML.

2. **The database code is tied to the business logic.** This is a problem of redundancy: If you rename your database tables or columns, you'll have to rewrite your SQL.

   By contrast, the Django/MVC approach encourages a single, abstracted data-access layer that's responsible for all data access. In Django's case, the data-access layer knows your database table and column names and lets you execute SQL queries via Python instead of writing SQL manually. This means, if database table names change, you can change it in a single place — your data-model definition — instead of in each SQL statement littered throughout your code.

3. **The URL is coupled to the code.** If this PHP file lives at `/foo/index.php`, it'll be executed for all requests to that address. But what if you want this same code to execute for requests to `/bar/` and `/baz/`? You'd have to set up some sort of includes or rewrite rules, and those get unmanageable quickly.