The Django Book

table of contents ◇ next »

Chapter 1: Introduction to Django

14

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:

1

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

1

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.

1

Let's break it down.

Django is a high-level Python Web framework…

1

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.

2

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.

5

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."

3

- **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).

2

- **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.

4

- **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.

1

- **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.

14

- **It gets out of your way**, neglecting to leave dirty stains on your application such as URLs that contain ".aspx" or ".php".

5

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

1

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>
```

3

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.

   4

   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.

   3

   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.

   7

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.

   1

   By contrast, Django decouples URLs from callback code, so you can change the URLs for a given piece of code.

4. **The database connection parameters and backend are hard-coded.** It's messy to have to specify connection information — the server, username and password — within this code, because that's configuration, not programming logic. Also, this example hard-codes the fact that the database engine is MySQL.

   1

   By contrast, Django has a single place for storing configuration, and the database-access layer is abstracted so that switching database servers (say, from MySQL to PostgreSQL) is easy.

15

What Django doesn't do

Of course, we want this book to be fair and balanced. With that in mind, we should be honest and outline what Django *doesn't* do:

- Feed your cat.
- Mind-read your project requirements and implement them on a carefully timed basis so as to fool your boss into thinking you're not really staying home to watch "The Price is Right."

On a more serious note, Django does not yet reverse the effects of global warming.

Why was Django developed?

Django is deeply rooted in the problems and solutions of the Real World. It wasn't created to be marketed and sold to developers, nor was it created as an academic exercise in somebody's spare time. It was built from Day One to solve daily problems for an industry-leading Web-development team.

It started in fall 2003, at — wait for it — a small-town newspaper in Lawrence, Kansas.

For one reason or another, The Lawrence Journal-World newspaper managed to attract a talented bunch of Web designers and developers in the early 2000s. The newspaper's Web operation, World Online, quickly turned into one of the most innovative newspaper Web operations in the world. Its three main sites, LJWorld.com (news), Lawrence.com (entertainment/music) and KUsports.com (college sports), began winning award after award in the online-journalism industry. Its innovations were many, including:

- The most in-depth local entertainment site in the world, Lawrence.com, which merges databases of local events, bands, restaurants, drink specials, downloadable songs and traditional-format news stories.
- A summer section of LJWorld.com that treated local Little League players like they were the New York Yankees — giving each team and league its own page, hooking into weather data to display forecasts for games, providing 360-degree panoramas of every playing field in the vicinity and alerting parents via cell-phone text messages when games were cancelled.
- Cell-phone game alerts for University of Kansas basketball and football games, which let fans get notified of scores and key stats during games, and a second system that used artificial-intelligence algorithms to let fans send plain-English text messages to the system to query the database ("how many points does giddens have" or "pts giddens").
- A deep database of all the college football and basketball stats you'd ever want, including a way to compare any two or more players or teams in the NCAA.
- Giving out blogs to community members and featuring community writing prominently — back before blogs were trendy.

Journalism pundits worldwide pointed to World Online as an example of the future of journalism. The New York Times did a front-page business-section story on the company; National Public Radio did a two-day series on it. World Online's head editor, Rob Curley, spoke nearly *weekly* at journalism conferences across the globe, showcasing World Online's innovative ideas and site features. In a bleak, old-fashioned industry resistant to change, World Online was a rare exception.

Much of World Online's success was due to the technology behind its sites, and the philosophy that computer programmers are just as important in creating quality 21st Century journalism as are journalists themselves.

This is why Django was developed: World Online's developers needed a framework for developing complex database-driven Web sites painlessly, easily and on journalism deadlines.

In fall 2003, World Online's two developers, Adrian Holovaty and Simon Willison, set about creating this framework. They decided to use Python, a language with which they'd recently fallen in love. After exploring (and being disappointed by) the available Python Web-programming libraries, they began creating Django.

Two years later, in summer 2005, after having developed Django to a point where it was efficiently powering most of World Online's sites, the World Online team, which now included Jacob Kaplan-Moss, decided it'd be a good idea to open-source the framework. That way, they could give back to the open-source community, get free improvements from outside developers, and generate some buzz for their commercial Django-powered content-management system, Ellington (http://www.ellingtoncms.com/). Django was open-sourced in July 2005 and quickly became popular.

Although Django is now an open-source project with contributors across the planet, the original World Online developers still provide central guidance for the framework's growth, and World Online contributes other important aspects such as employee time, marketing materials and hosting/bandwidth for the framework's Web site (http://www.djangoproject.com/).

Who uses Django?

Web developers around the world use Django. Some specific examples:

- World Online, of course, continues to use Django for all its Web sites, both internal and for commercial clients. Some of its Django-powered sites are:
  - http://www.ljworld.com/
  - http://www.lawrence.com/
  - http://www.6newslawrence.com/
  - http://www.visitlawrence.com/
  - http://www.lawrencechamber.com/
  - http://www2.kusports.com/stats/
- The Washington Post's Web site, washingtonpost.com, uses Django for database projects and various bits of functionality across the site. Some examples:
  - The U.S. Congress votes database, http://projects.washingtonpost.com/congress/
  - The staff directory and functionality that lets readers contact reporters, appearing as links on most article pages.
  - Faces of the Fallen, http://projects.washingtonpost.com/fallen/
- Chicagocrime.org, a freely browsable database of crime reported in Chicago and one of the original Google Maps mashups, was developed in Django.
- Tabblo.com, an innovative photo-sharing site, uses Django. The site lets you piece together your photos to create photo pages that tell stories.
- Texasgigs.com, a local music site in Dallas, Texas, was written with Django.
- Grono.net, a Polish social-networking site, started replacing its Java code with Django. It found that Django not only was faster (and more fun) to develop in — it performed better than Java and required less hardware.
- Traincheck.com was developed in Django. The site lets you send text-messages from your cell phone to get subway train schedules for your immediate location.

An up-to-date list of dozens of sites that use Django is located at http://code.djangoproject.com/wiki/DjangoPoweredSites

About this book

The goal of this book is to explain all the things Django does — and to make you an expert at using it.

By reading this book, you'll learn the skills needed to develop powerful Web sites quickly, with code that's clean and easy to maintain.

We're glad you're here!

table of contents ◇ next »