

Archive for the "Pylons" category

Django vs Pylons

admin · Sunday, November 29th, 2009

This is an attempt to make a non-biased comparison between Django and Pylons. I did a brief comparison of the two at work since we needed a web framework for our new project.

We are making a web application that will serve as a frontend to a bank database. The application is going to allow bank employees to insert and track interbank loans and their repayments. The database is actually abstracted behind a layer built with Twisted, so the web application will only use XMLRPC calls to communicate with the database, hence no ORM.

As of the moment of writing, Django is in version 1.1 and Pylons is in version 0.9.7. Both of the frameworks are mature and tested in production in some major websites. Django has [Washington Post](#) and pylons has [Reddit](#).

The frameworks are quite different in philosophy, although both follow the same MVC paradigm. Django has more magic and less code, while pylons has more code and less magic. Pylons is essentially a bare-bones wrapper around the WSGI specification that uses 3rd party modules for templating, database interaction, routing and just about anything else, while Django is aimed towards rapid development of web applications and has everything packed inside of it - it's own template system, routing and ORM. This allows Django to establish high reusability for code between different projects. On the other hand, the developer is limited to one ORM and templating system.

Let's go point by point. These are not all the relevant features in a web framework by any means, but they should give the reader a general overview of the differences and the strengths of the frameworks in the most important areas.

Structure

Pylons reminds on Ruby on Rails here. It keeps all the controllers in one directory, all the models in another directory and all the templates in a third directory. Django is made to be more "pluggable". In Django you create apps that are self-contained, meaning they have their controller, models and possibly templates in one python package that you can easily pull out of the project and put it in another one. The "pluggability" of Django comes from the fact that you're using the same components in every project.

Pylons on the other hand is made to be more "swappable", meaning that you can easily switch a core functionality module, like the templating engine, with another one. This however, limits the reusability of your applications between projects.

I like Django's approach better because the files seem much better organized, but there is a slight disadvantage - the files have the same names. For example: if you have 4 apps, each of them contains a views.py file and it's a bit confusing when you got the mall open at once in an editor. But that's really nitpicking.

Documentation

Being a collection of 3rd party modules, Pylons' documentation is scattered all over the internet. You will need to read the SQLAlchemy docs for database, Mako templates' docs for templating, Routes' docs for routing, Paste's docs for general project tasks, Babel's docs for internationalization, etc. Django on the other hand has very extensive documentation on one place, and I must say it's much more comprehensive. There is one book for Pylons, but there are a dozen books for Django. This doesn't say much, except that Django is clearly more popular. This is a clear advantage to Django.

Community and Development

The Django community is huge compared to Pylon's. On [Stack Overflow](#) there are 3600 questions tagged "django" and only 100 tagged "pylons". Google trends shows similar results (around 15 times more for Django). I can't really talk about Pylon's community, but Django's is really friendly and helpful. Django also seems to be in much more active development than Pylons, but that's maybe because there is more work to be done. Pylons depends more on the development of the 3rd party modules it uses.

Learning Curve

This is somewhat dependent on the previous two points. A larger community, more code samples and better documentations works a

long way towards flattening the learning curve. On the other hand we should consider our previous experience with the two frameworks. If you already are familiar with SQLAlchemy and the WSGI spec, then Pylons requires less work to learn. Otherwise if you have experience with just python, then Django would be easier to grasp, only because of the availability of resources.

Coding required

Django, in principle, requires less code to do the same task. Less code means more magic and that is not always the best thing. Django was developed in a newspaper publishing environment where deliveries were really time-constrained and the authors created some very impressive tools to enable them to add website features quickly. For instance, Django comes with an automatic administration app that doesn't require you to write a single line of code. If you, however, strip Django of its ORM - much of this magic goes away.

Templates

Pylons the **Mako** templating language by default, but you can also use others, like **Jinja**. Jinja (as Jon pointed out) is based on Django templates and I find it much cleaner than Mako because it restricts you from running arbitrary python code inside template files (as does Django).

People tend to say that Mako templates are more powerful, but then PHP is also very powerful as a templating language and it always produces illegible tag soup. It's very tempting to start assigning and calculating values inside templates, especially when you're really tired.

Django templates and Jinja give you just enough power to get the job done and not a watt more :) They have only two special constructs - double braces for outputting values and brace followed by percent sign for invoking template tags. In Mako you have: "<%" for opening python blocks, "\${variable}" for variable printing, "%" for control structures, "<%!" for module-level blocks and "##" for comments.

Update: I just had a look at Jinja templates and it looks like they offer some interesting tools on top of Django templates, like simple expressions (e.g. `{{2+2}}`) inside the template tags. The only way to do some of the stuff that Jinja does out of the box with Django is by writing your own template tag. There is a nice post on why you should use Jinja, even with Django [here](#).

Anyway, you have the choice with Pylons.

Routing

Pylons uses the **Routes** library for routing. Routes is a clone of Ruby on Rails routing which is also inferior to Django's routing. Routes is implicit in a way that you define a route. For example `"{controller}/{action}"` routes to the 'action' method of the 'controller' class in your project. There is a possibility to use regular expressions but they come as additional parameters to the route declaration. Django's routes pair a regex to an explicitly specified view. Also they are much less verbose. As far as I can see (I may be wrong) Routes keeps all the routing declarations in a single file, while in Django you can chop off the first part of a URL and pass the rest to another url regex thus decoupling project apps.

Don't get me wrong here; both routing systems are capable, but Django's is just more natural and more pythonic.

ORM and Database

I don't have any experience with **SQLAlchemy**, but from what I read around it sounds like it's better than Django's ORM. It's the only option if you intend to use SQL Server anyway. Django was developed before SQLAlchemy was around so it uses it's own ORM, which is quite nice really. Coming from PHP, it's light years ahead of anything PHP has (take a look at **Propel**).

Since I can't say too much on this particular point, here's a few links that can fill you in:

<http://adam.gomaa.us/blog/2007/aug/1/five-things-i-hate-about-django/>

<http://reliablybroken.com/b/2008/06/choosing-sqlalchemy-over-django/>

<http://outatime.wordpress.com/2008/03/26/django-sqlalchemy/>

<http://jmoiron.net/blog/about-sqlalchemy-and-djangos-orm/>

Choices, Choices

In the end we went with Pylons. Our problem is not yet completely defined so we have to leave a lot of room for maneuvers. We don't know if the application will be just a single form or it will grow to replace the current desktop banking application. It's not going to have a database backend, no classic administration panel and no classic authentication. These are the areas where Django shines. If you leave them behind, the advantages that Django has fade out. Pylons however, with it's flexibility is a perfect choice for this kind of web applications.

On the other hand, if I had a standard website project I would go with Django all the way. It's really the best at rapid development of web applications and yes, the magic feels good. Sometimes you really don't care about what you are able to do **with** your framework, rather for what your framework can do **for** you. And Django can do an awful lot.

If you ask me, the biggest strength is the biggest weakness of both frameworks.

Pylons is a collection of 3rd party libraries. This means that you always get the latest and greatest of the Python world. But this also means that the community is much less compact and there is more manual work. Also, there is the problem of choice - you need to spend time and think about which component you'll use for each task. You don't want to end up with several projects all using different templating engines.

Django has the benefits of a huge, unified community, but it's weakness is that it's components are coupled and interdependent. [Here's a really good talk about the Django problems](#)

Posted in [Django](#), [Pylons](#), [Python](#) | [6 Comments »](#)