Django vs other Python web frameworks?

Asked 15 years, 8 months ago Modified 9 years, 1 month ago

Viewed 35k times

61

I've pretty much tried every Python web framework that exists, and it took me a long time to realize there wasn't a silver bullet framework, each had its own advantages and disadvantages. I started out with Snakelets and heartily enjoyed being able to control almost everything at a lower level without much fuss, but then I discovered TurboGears and I have been using it (1.x) ever since. Tools like Catwalk and the web console are invaluable to me.

But with TurboGears 2 coming out which brings WSGI support, and after reading up on the religious debates between the Django and WSGI camps, I'm really torn between ***"doing it the right way"***, e.g., learning WSGI, spending valuable time writing functionality that already exists in Django and other full-stack frameworks, as opposed to using Django or some high-level framework that does everything for me. The downsides with the latter that I can see are pretty obvious:

1. I'm not learning anything in the process
2. If I ever need to do anything lower level it's going to be a pain

3. The overhead required for just a basic site which uses authentication is insane. (IMO)

So, I guess my question is, which is the better choice, or is it just a matter of opinion, and should I suck it up and use Django if it achieves what I want with minimal fuss (I want authentication and a CRUD interface to my database)? I tried Werkzeug, Glashammer, and friends, but AuthKit and Repoze scared me off, as well as the number of steps involved to just setup basic authentication. I looked at Pylons, but the documentation seems lacking, and when referencing simple features like authentication or a CRUD interface, various wiki pages and documentation seemed to contradict each other, with different hacks for versions and such.

Thanks to S. Lott for pointing out that I wasn't clear enough. My question is: which of the following is worthwhile in the long run, but not painful in the short (e.g., some sort of middle ground, anyone?) - Learn WSGI, or stick with a "batteries-included" framework? If the latter, I would appreciate a suggestion as to whether I should give Django another try, stick with TurboGears 1.x, or venture into some other framework.

Also, I have tried CherryPy, but couldn't seem to find a good enough CRUD application that I could plop in and use right away.

python **django** **wsgi** **web-frameworks** **turbogears**

Share

Improve this question

Follow

edited Mar 31, 2009 at 18:08

community wiki
2 revs
miniman

Could you clarify your question. "which is the better choice" among what options? Django vs. TurboGears 1 vs. TurboGears 2? Please clarify so we know what you're really asking and what help you really need. – S.Lott Mar 31, 2009 at 17:43

13 Answers

Sorted by: Highest score (default)

55

> the religious debates between the Django and WSGI camps

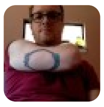
It would seem as though you're a tad bit confused about what WSGI is and what Django is. Saying that Django and WSGI are competing is a bit like saying that C and SQL are competing: you're comparing apples and oranges.

Django is a framework, WSGI is a protocol (which is supported by Django) for how the server interacts with the framework. Most importantly, learning to use WSGI directly is a bit like learning assembly. It's a great learning experience, but it's not really something you should do for production code (nor was it intended to be).

At any rate, my advice is to figure it out for yourself. Most frameworks have a "make a wiki/blog/poll in an hour" type exercise. Spend a little time with each one and figure out which one you like best. After all, how can you decide between different frameworks if you're not willing to try them out?

Share Improve this answer Follow

answered Mar 31, 2009 at 22:25



Jason Baker
198k ● 138 ● 382 ● 520

21

I'd say you're being a bit too pessimistic about "not learning anything" using Django or a similar full-stack framework, and underestimating the value of documentation and a large community. Even with Django there's still a considerable learning curve; and if it doesn't do everything you want, it's not like the framework code is impenetrable.

Some personal experience: I spent years, on and off, messing around with Twisted/Nevow, TurboGears and a few other Python web frameworks. I never finished anything because the framework code was perpetually unfinished and being rewritten underneath me, the documentation was often nonexistent or wrong and the only viable support was via IRC (where I often got great advice, but felt like I was imposing if I asked too many questions).

By comparison, in the past couple of years I've knocked off a few sites with Django. Unlike my previous experience, they're actually deployed and running. The Django development process may be slow and careful, but it results in much less bitrot and deprecation, and documentation that is actually helpful.

HTTP authentication support for Django finally went in a few weeks ago, if that's what you're referring to in #3.

Share Improve this answer Follow

answered Mar 31, 2009 at 18:11

Nicholas Riley
44.3k ● 6 ● 104 ● 125

How do you compare repoze.bfg in terms of documentation, support, finishing-touch, etc..? – Sridhar Ratnakumar Oct 4, 2009 at 1:22

16

I suggest taking another look at TG2. I think people have failed to notice some of the strides that have been made since the last version. Aside from the growing WSGI stack of utilities available there are quite a few TG2-specific items to consider. Here are a couple of highlights:

TurboGears Administration System - This CRUD interface to your database is fully customizable using a declarative config class. It is also integrated with Dojo to give you infinitely scrollable tables. Server side validation is also automated. The admin interface uses RESTful urls and

HTTP verbs which means it would be easy to connect to programatically using industry standards.

CrudRestController/RestController - TurboGears provides a structured way to handle services in your controller. Providing you the ability to use standardized HTTP verbs simply by extending our RestController. Combine Sprox with CrudRestController, and you can put crud anywhere in your application with fully-customizable autogenerated forms. TurboGears now supports mime-types as file extensions in the url, so you can have your controller render .json and .xml with the same interface it uses to render html (returning a dictionary from a controller)

If you click the links you will see that we have a new set of documentation built with sphinx which is more extensive than the docs of the past.

With the best web server, ORM, and template system(s) (pick your own) under the hood, it's easy to see why TG makes sense for people who want to get going quickly, and still have scalability as their site grows.

TurboGears is often seen as trying to hit a moving target, but we are consistent about releases, which means you won't have to worry about working out of the trunk to get the latest features you need. Coming to the future: more TurboGears extensions that will allow your application to grow functionality with the ease of paster commands.

answered Mar 31, 2009 at 21:54

Share Improve this answer

Follow

community wiki
percious

2 technically I had no problem with Turbogears. But I found the community elitist and less helpful to newcomers than Rails / Django / web2py. – hoju Nov 26, 2011 at 1:21

11

Your question seems to be "is it worth learning WSGI and doing everything yourself," or using a "full stack framework that does everything for you."

I'd say that's a false dichotomy and there's an obvious third way. TurboGears 2 tries to provide a smooth path from a "do everything for you" style framework up to an understanding of WSGI middleware, and an ability to customize almost every aspect of the framework to suit your application's needs.

We may not be successful in every place at every level, but particularly if you've already got some TurboGears 1 experience I think the TG2 learning curve will be very, very easy at first and you'll have the ability to go deeper exactly when you need it.

To address your particular issues:

- We provide an authorization system out of the box that matches the one you're used to from TG1.

- We provide an out of the box "django admin" like interface called the tgext.admin, which works great with dojo to make a fancy spreadsheet like interface the default.

I'd also like to address a couple of the other options that are out there and talk a little bit about the benifits.

- **CherryPy.** I think CherryPy is a great webserver and a nice minimalistic web-framework. It's not based on WSGI internally but has good WSGI support although it will not provide you with the "full stack" experience. But for custom setups that need to be both fast and aren't particularly suited to the defaults provided by Django or TurboGears, it's a great solution.
- **Django.** I think Django is a very nice, tigtly integrated system for developing websites. If your application and style of working fits well within it's standard setup it can be fantastic. If however you need to tune your DB usage, replace the template language, use a different user authorization model or otherwise do things differently you may very likely find yourself fighting the framework.
- **Pylons** Pylons like CherryPy is a great minimalistic web-framework. Unlike CherryPy it's WSGI enabled through the whole system and provides some sane defaults like SQLAlchemy and Mako that can help you scale well. The new official docs are of much

better quality than the old wiki docs which are what you seem to have looked at.

Share Improve this answer Follow

edited Nov 16, 2012 at 21:46

community wiki
3 revs, 3 users 85%
Mark Ramm

7

Have you taken a look at CherryPy. It is minimalistic, yet efficient and simple. It is low level enough for not it to get in they way, but high enough to hide complexity. If I remember well, TurboGears was built on it.

With CherryPy, you have the choice of much everything. (Template framework, ORM if wanted, back-end, etc.)

Share Improve this answer Follow

answered Mar 31, 2009 at 17:46

Martin
6,022 ● 5 ● 32 ● 47

7

> Learn WSGI

WSGI is absurdly simple.. It's basically a function that looks like..

```
def application(environ, start_response) pass
```

The function is called when an HTTP request is received. `environ` contains various data (like the request URI etc etc), `start_response` is a callable function, used to set headers.

The returned value is the body of the website.

def application(environ, start_response): start_response("200 OK", []) return "..."

That's all there is to it, really.. It's not a framework, but more a protocol for web-frameworks to use..

For creating sites, using WSGI is *not* the "right way" - using existing frameworks is.. but, if you are writing a Python web-framework then using WSGI is absolutely the right way..

Which framework you use (CherryPy, Django, TurboGears etc) is basically personal preference.. Play around in each, see which you like the most, then use it.. There is a StackOverflow question (with a great answer) about this, "Recommendation for straight-forward python frameworks"

Share Improve this answer Follow

edited May 23, 2017 at 11:54
Community Bot
1 ● 1

answered Mar 31, 2009 at 22:44
dbr
169k ● 69 ● 283 ● 347

2

Have you checked out web2py? After recently evaluating many Python web frameworks recently I've decided to adopt this one. Also check out Google App Engine if you haven't already.

Share Improve this answer Follow

answered Mar 31, 2009 at 18:12

greg
29 ● 1

2

I'd say the correct answer depends on what you actually want and need, as what will be worthwhile in the long run depends on what you'll need in the long run. If your goal is to get applications deployed ASAP then the 'simpler' route, ie. Django, is surely the way to go. The value of a well-tested and well-documented system that exactly what you want can't be underestimated.

On the other hand if you have time to learn a variety of new things which may apply in other domains and want to have the widest scope for customisation then something like Turbogears is superior. Turbogears gives you maximum flexibility but you *will* have to spend a lot of time reading external docs for things like Repoze, SQLAlchemy, and Genshi to get anything useful done with it. The TG2 docs are deliberately less detailed than the TG1 docs in some cases because it's considered that the external docs are better than they used to be. Whether this sort of thing is an obstacle or an investment depends on your own requirements.

1

Django is definitely worth learning, and sounds like it will fit your purposes. The admin interface it comes with is easy to get up and running, and it does use authentication.

As for "anything lower level", if you mean sql, it is entirely possible to shove sql into you queries with the extra keyword. Stylistically, you always try to avoid that as much as possible.

As for "not learning anything"...the real question is whether your preference is to be primarily learning something lower-level or higher-level, which is hardly a question anyone here can answer for you.

Pylons seems a great tool for me:

1

- a real web framework (CherryPy is just a web server),
- small code base - reuse of other projects,
- written entirely with WSGI in mind, based on Paste,
- allows you to code the app right away and touch the low level bits if it's necessary,

I've used CherryPy and TurboGears and look at many other frameworks but none of them were so light and productive as Pylons is. Check the presentation at Google.

Share Improve this answer Follow

answered May 29, 2009 at 14:09

community wiki
lispmachine

0

I'm a TurboGears fan, and this is exactly the reason why: a very nice trade-off between control and doing things right vs. easy.

You'll have to make up your own mind of course. Maybe you'd prefer to learn less, maybe more. Maybe the areas that I like knowledge/control (database for example), you couldn't care less about. And don't misunderstand. I'm not characterizing any frameworks as necessarily hard or wrong. It's just my subjective judgment.

Also I would recommend TurboGears 2 if at all possible. When it comes out, I think it will be much better than 1.0 in terms of what it has selected for defaults (genshi, pylons, SqlAlchemy)

Share Improve this answer Follow

answered Mar 31, 2009 at 19:11

community wiki
Dan

0

I would suggest for TurboGears 2. They have done a fantastic job of integrating best of Python world.

WSGI: Assuming you are developing moderately complex projects/ business solutions in TG2 or some other framework say Grok. Even though these frameworks supports WSGI does that mean one who is using these frameworks have to learn WSGI? In most cases answer is No. I mean it's good have this knowledge no doubt.

WSGI knowledge is probably is more useful in cases like

- you want to use some middleware or some other component which is not provided as part of the standard stack for eg. Authkit with TG or Grok without ZODB.
- you are doing some integration.

CherryPy is good but think of handling your database commits/rollbacks at the end of transactions, exposing json, validations in such cases TG, Django like frameworks do it all for you.

Share Improve this answer Follow

answered Mar 31, 2009 at 19:46

community wiki
Shekhar

2 CherryPy is more pythonic, if you follow the rule from the zen(python.org/dev/peps/pep-0020): Explicit is better than implicit., Simple is better than complex. I am not a fan of magic happening behind the scenes like commits and rollbacks. – Martin Mar 31, 2009 at 20:14

-1

Web2py is the secret sauce here. Don't miss checking it out.

Share Improve this answer Follow

answered Nov 12, 2015 at 1:36

community wiki
John Finch