Django Packages Documentation

Release

Audrey Roy, Daniel Greenfeld and contributors

Sep 27, 2017

Contents

Django Packages solves the problem in the programming community of being able to easily identify good apps, frameworks, and packages. Ever want to know which is the most popular or well supported Python httplib replacement, web framework, or api tool? Django Packages solves that problem for you!

It does this by storing information on packages fetched from public APIs provided by PyPI, Github, and BitBucket, then provides extremely useful comparison tools for them.

CHAPTER 1

Contributing to Django Packages

1. Follow the installation instructions!
2. Follow the contributing instructions!

Contents:

Introduction

Ever want to know which is the most popular or well supported Python httplib replacement, web framework, or api tool? Django Packages solves that problem for you! Django Packages allows you to easily identify good apps, frameworks, and packages.

Django Packages stores information on fetched packages and provides easy comparison tools for them. Public APIs include PyPI, Github, and BitBucket.

The Site

A current example is live: http://www.djangopackages.com

Grids!

Grids let you compare packages. A grid comes with default comparison items and you can add features to get a more specific. We think comparison grids are an improvement over traditional tagging system because specificity helps make informed decisions.

Categories of Packages

The fixtures provide four categories: apps, frameworks, projects, and utilities.

What repo sites are supported?

- Github
- Bitbucket

License

Installation

Do everything listed in this section to get your site up and running locally. If you run into problems, see the Troubleshooting section.

Pre-requisites

Mac OS X 10.6

Download and install setuptools from http://pypi.python.org/pypi/setuptools. Setuptools gives you easy_install. Then run the following commands:

```
easy_install pip
pip install virtualenv
brew install libmemcached
```

Ubuntu (10+ / Lucid or Higher)

Install the following:

```
sudo apt-get install python-setuptools python-dev libpq-dev
sudo easy_install pip
sudo pip install virtualenv
```

Windows 7

Download and install Python 2.6 or 2.7 using the Windows 32-bit installer from http://www.python.org/download/. Even if you're on a 64-bit system, 32-bit is recommended (Michael Foord told me this).

Download and install setuptools from http://pypi.python.org/pypi/setuptools. Setuptools gives you easy_install.

Install MinGW from http://www.mingw.org/. Add the bin/ directory of your MinGW installation to your PATH environment variable (under Control Panel > System > Advanced system settings > Environment variables).

Create or open C:\Python26\Lib\distutils\distutils.cfg (Note: this may be inside the Python27 directory if you're using Python 2.7). Add the following lines to the bottom of the file:

```
[build]
compiler=mingw32
```

Open up a command prompt. Install pip and virtualenv:

```
easy_install pip
pip install virtualenv
```

Other operating systems (including various Linux flavors)

We don't provide instructions for these, but you should be able to figure things out from the provided instructions. See *FAQ*.

Main instructions

These instructions install Django Packages on your computer, using PostgreSQL and sample data.

Git clone the project and install requirements

Create a virtualenv, activate it, git clone the Django Packages project, and install its requirements:

```
cd <installation-directory>
virtualenv env-oc
source env-oc/bin/activate
git clone git@github.com:opencomparison/opencomparison.git opencomparison
cd opencomparison
pip install -r requirements.txt
```

Set up server specific settings

Don't change `settings/base.py`. Instead extend it as you see in `settings/heroku.py`. In the new file make the following specifications:

Add a Google Analytics code if you have one:

```
URCHIN_ID = "UA-YOURID123-1"
```

Setup your email settings:

```
DEFAULT_FROM_EMAIL = 'Your Name <me@mydomain.com>'
EMAIL_SUBJECT_PREFIX = '[Your Site Name] '
```

Set up your PostgreSQL database

Set up PostgreSQL and create a database:

```
createdb oc
```

For more info, see *PostgreSQL setup instructions for new contributors*.

Set up the database tables:

```
python manage.py syncdb --no-input
python manage.py migrate
```

Note: This is optional. You can load some base data for development usage (i.e. not in production):

```
python manage.py load_dev_data
```

Grids will not be listed on the homepage, but they are available on /grids/

Load the site in your browser

Run the development server:

```
python manage.py runserver
```

Then point your browser to http://127.0.0.1:8000

Give yourself an admin account on the site

Create a Django superuser for yourself, replacing joe with your username/email:

```
python manage.py createsuperuser --username=joe --email=joe@example.com
```

And then login into the admin interface (/admin/) and create a profile for your user filling all the fields with any data.

Deployments

For various providers and methods including Heroku, DotCloud, et al.

Contents:

Heroku

Quick and Easy Heroku Deployment

This is meant for setting up test and QA servers, not full deployments.

If you have Heroku Toolbelt installed and are on Mac OS X or Linux, you can run the following from your repo:

```
make createsite
```

Real Heroku Deployment

Deploying to heroku:

```
heroku create --stack cedar
git push heroku master
heroku addons:add heroku-shared-postgresql
heroku addons:add memcachier:dev
heroku addons:add sendgrid:starter
heroku addons:add scheduler:standard
heroku pg:promote HEROKU_SHARED_POSTGRESQL_GOLD
heroku pg:psql HEROKU_SHARED_POSTGRESQL_GOLD
\i django_oc.sql
```

Chron Jobs that need to be set up

Sample:

```
python manage.py pypi_updater --settings=settings.heroku
python manage.py repo_updater --settings=settings.heroku
python manage.py searchv2_build --settings=settings.heroku
```

TODO: Email admins with the log

Custom settings that need to be added

Do the following:

```
heroku config:add SECRET_KEY=<random-key>
heroku config:add GITHUB_API_SECRET=CUSTOM
heroku config:add GITHUB_APP_ID=CUSTOM
heroku config:add SITE_TITLE=Django Packages
heroku config:add FRAMEWORK_TITLE=Django
heroku config:add AWS_ACCESS_KEY_ID=CUSTOM
heroku config:add AWS_SECRET_ACCESS_KEY=CUSTOM
heroku config:add AWS_STORAGE_BUCKET_NAME=CUSTOM
```

Troubleshooting

How come no module named abc?

If you're getting something like "ImportError: No module named abc", you probably don't have all the required packages installed. Try:

```
pip install -r requirements/project.txt
```

No module named psycopg2

If you're getting something like "ImproperlyConfigured: Error loading psycopg2 module: No module named psycopg2" while accessing the website, you need to install the psycopg2 module. It has recently been added to requirements/project.txt (the line that says "psycopg2==2.4"). Try:

```
pip install -r requirements/project.txt
```

If you're getting an error like "Error: pg_config executable not found." while installing the module, you need the PostgreSQL development package. On Ubuntu, do:

```
sudo apt-get install libpq-dev
```

I can't get it to work in buildout!

We don't support buildout. See the faq.

bz2 not found

Install the appropriate systemwide package. For example, on Ubuntu do:

    sudo apt-get install libbz2-dev

If this doesn't work, please let us know (create an issue at http://github.com/opencomparison/opencomparison/issues)

fatal error: 'libmemcached/memcached.h' file not found

if you are getting something like ./_pylibmcmodule.h:42:10: fatal error: 'libmemcached/memcached.h' file not found. Then you need to install libmemcached:

```
brew install libmemcached
```

Other problems

Don't give up! Submit problems to http://github.com/opencomparison/opencomparison/issues. And don't forget:

1. Be polite! We are all volunteers.
2. Spend the time to learn Github markup

FAQ

General

How did Django Packages get started?

- In 2010 We realized there was no effective method for finding apps in the Django community.
- After launch we realized it might be good to use the same software system for other package sets.

Are there any Case Studies?

- http://pycon.blip.tv/file/4878766
- http://www.slideshare.net/pydanny/django-packages-a-case-study

How can I contribute?

Read the page on contributions.

How can I add a listing for a new Package or an entirely new Grid?

- Go the Home page, https://www.djangopackages.com/
- Go to the left side section called "Add packages and grids".
- Click the appropriate button, where a package is a program and a grid is a category.

What browsers does Django Packages support?

We do formal tests on Chrome, Safari and Firefox.

How hard is it to add support for a new repo?

We've done a lot of work to make it as straightforward as possible. At PyCon 2011 we launched our formal Repo Handler API.

Installation

How come you don't support buildout?

We have a very successful installation story for development and production hosting using virtualenv. While buildout is a wonderful tool we simply don't want to spend the time supporting two installation methods. Therefore:

- Don't do it.
- We won't accept pull requests for it.

Why don't you have install instructions for BSD? Or Debian? Or Windows XP?

If you are using something else besides Ubuntu, Mac OS X 10.6+, or Windows 7, you obviously have mad skills. We have a very successful installation story for development on three very common operating systems and production hosting is assumed to be on Ubuntu or any of the major PaaS vendors. Trying to support more than those operating systems is a HUGE amount of time taken away from making improvements - especially since the core developers insist on testing everything themselves.

What happened to the fixtures?

The effort to support databases besides PostGreSQL was hampered for long time, all caused by a third party package we're not going to identify that caused grief in the use of fixtures. This was a significant issue in Django Packages, and used up a lot of development cycles.

So we use a **Mock** system of creating sample data in our tests and for running a development version of the site. To create some development data, just run:

```
python manage.py load_dev_data
```

Unsupported Repo Hosting Services

Django Packages supports GitHub and BitBucket. Here is some information about other repo hosting services.

Google Project Hosting

How come you don't support google project hosting?

They don't have an API. We've filed ticket #5088 and we hope the nice people there can close it in the near future. Google is part of the open source world and we would love to support projects using their hosting services.

What about the Google Project Hosting Issue API?

Django Packages doesn't track a project's tickets/issues.

What about just screen scraping their site?

Too brittle for our tastes. The Google Project hosting site uses a lot of JavaScript and AJAX to deliver content. Besides, we would like to think our fellow developers at Google will provide us with a really awesome, well-documented, stable API.

Launchpad

In 2011, when we provided support, their API client involved 5 MB of external dependencies, which is just plain silly for a RESTful API system. We also had a large number of failures by third-party contributors trying to work with their toolchain. We thought about creating a urlib/urllib2 (later requests) powered custom API client, but the demand for Launchpad support is too low to justify the work.

Since then, we've pulled all the Launchpad specific code out of Django Packages.

If you want launchpad support, we welcome pull requests.

Sourceforge

In 2011 we tried to provide support but their API was not adequate for our needs. Since then we've not had a request for Sourceforge support.

If you want Sourceforge support, we know their API has improved and we welcome pull requests.

Gitorious

We've had the odd request for Gitorious support. Their API is adequate and we welcome pull requests.

Settings

How to customize the settings to suit your needs.

ADMIN_URL_BASE (Default: r"^admin/")

Used to control the URL for the admin in production.

FRAMEWORK_TITLE (Default: "Django")

Used to create the name of the site.

PACKAGINATOR_SEARCH_PREFIX (Default: "django")

Autocomplete searches for something like 'forms' was problematic because so many packages start with 'django'. This prefix is accommodated in searches to prevent this sort of problem.

example:

```
PACKAGINATOR_SEARCH_PREFIX = 'django'
```

PACKAGINATOR_HELP_TEXT (Default: Included in settings.py)

Used in the Package add/edit form in both the admin and the UI, these are assigned to model form help text arguments. Takes a dict of the following items:

Example (also the default):

```
PACKAGINATOR_HELP_TEXT = {
    "REPO_URL" : "Enter your project repo hosting URL here.<br />Example: https://
↪bitbucket.org/ubernostrum/django-registration",
    "PYPI_URL" : "<strong>Leave this blank if this package does not have a PyPI␣
↪release.</strong><br />What PyPI uses to index your package. <br />Example: django-
↪registration"
}
```

Permissions Settings

Django Packages provides several ways to control who can make what changes to things like packages, features, and grids. By default, a Django Packages project is open to contributions from any registered user. If a given project would like more control over this, there are two settings that can be used.

RESTRICT_PACKAGE_EDITORS RESTRICT_GRID_EDITORS

If these are not set, the assumption is that you do not want to restrict editing.

If set to True, a user must have permission to add or edit the given object. These permissions are set in the Django admin, and can be applied per user, or per group.

Settings that are on by default

By default registered users can do the following:

Packages

- Can add package
- Can change package

Grids

- Can add Package
- Can change Package
- Can add feature
- Can change feature
- Can change element

In the default condition, only super users or those with permission can delete.

Testing permissions in templates

A context processor will add the user profile to every template context, the profile model also handles checking for permissions:

```
{% if profile.can_edit_package %}
    <edit package UI here>
{% endif %}
```

The follow properties can be used in templates:

- can_add_package
- can_edit_package
- can_edit_grid
- can_add_grid
- can_add_grid_feature
- can_edit_grid_feature
- can_delete_grid_feature
- can_add_grid_package
- can_delete_grid_package
- can_edit_grid_element

Testing Instructions

Running the test suite

To run all of the Django Packages tests:

```
python manage.py test --settings.test
```

To run tests for a particular Django Packages app, for example the feeds app:

```
python manage.py test feeds --settings.test
```

Management Commands

package_updater

You can update all the packages with the following command:

```
python manage.py package_updater
```

Warning: This can take a long, long time.

PyPI Issues

You may ask why the PyPI code is a bit odd in places. PyPI is an organically grown project and uses its own custom designed framework rather than the dominant frameworks that existed during its inception (these being Pylons, Django, TurboGears, and web.py). Because of this you get things like the API having in its package_releases() method an explicit license field that has been replaced by the less explicit list column in the very generic classifiers field. So we have to parse things like this to get a particular package's license:

```
['Development Status :: 5 - Production/Stable', 'Environment :: Web Environment',
'Framework :: Django', 'Intended Audience :: Developers', 'License :: OSI Approved
:: BSD License', 'Operating System :: OS Independent', 'Programming Language ::
Python', 'Topic :: Internet :: WWW/HTTP', 'Topic :: Internet :: WWW/HTTP ::
Dynamic Content', 'Topic :: Internet :: WWW/HTTP :: WSGI', 'Topic :: Software
Development :: Libraries :: Application Frameworks', 'Topic :: Software
Development :: Libraries :: Python Modules']
```

The specification is here and this part of it just makes no sense to me:

```
http://docs.python.org/distutils/setupscript.html#additional-meta-data
```

Team

Project Leads

- Audrey Roy <audreyr@gmail.com> (@audreyr)
- Daniel Greenfeld <pydanny@gmail.com> (@pydanny)

Core Developer at Server Move 2012

- Randall Degges

Core Developers at DjangoCon 2011

- James Punteney
- Mike Johnson
- Taylor Mitchell

Core Developer at DataMigrationCon 2011

- Katharine Jarmul

Core Developers at PyCon 2011

- Gisle Aas
- Nate Aune
- Szilveszter Farkas

Core Developers at DjangoCon 2011

- James Punteney
- Jonas Obrist
- Taavi Taijala

Direct Contributors

- Aaron Kavlie
- Adam Saegebarth
- Alex Robbins
- Andrii Kurinny
- AnneTheAgile
- Baptiste Mispelon
- Brian Ball
- Bryan Weingarten
- Chris Adams
- Christopher Clark
- David Peters
- Dougal Matthews (@d0ugal)
- Emmanuelle Delescolle (@nanuxbe)
- Eric Spunagle
- Evgeny Fadeev
- Fábio C. Barrionuevo da Luz
- Flaviu Simihaian
- George Dorn
- Gisle Aas (Repo Man)
- idealatom
- Ilian Iliev (@IlianIliev)

- Jacob Burch
- James Pacileo
- James Punteney
- Jeff Schenck
- Jim Allman
- John M. Camara
- Jonas Obrist
- jrothenbuhler
- Kenneth Love
- Kenneth Reitz
- @kerridge0
- Kulbir Singh
- Marc Tamlyn
- Marcin Lulek
- Mike Fiedler
- Mike Johnson
- Nate Aune
- Nolan Brubaker
- PA Parent
- Preston Holmes
- Randall Degges
- Skot Carruth
- Stuart Powers
- Szilveszter Farkas (Repo Man)
- Taavi Taijala
- Taylor Mitchell
- Tom Brander
- Yony Narlock
- Vasja Volin

Other Contributors

- The entire Python community for providing us the tools we needed to build this thing.

Contributing

Setup

Fork on GitHub

Before you do anything else, login/signup on GitHub and fork Django Packages from the GitHub project.

Clone your fork locally

If you have git-scm installed, you now clone your git repo using the following command-line argument where <my-github-name> is your account name on GitHub:

```
git clone git@github.com:<my-github-name>/opencomparison.git
```

Installing Django Packages

Follow our detailed installation instructions. Please record any difficulties you have and share them with the Django Packages community via our issue tracker.

Issues!

The list of outstanding Django Packages feature requests and bugs can be found on our on our GitHub issue tracker. Pick an unassigned issue that you think you can accomplish, add a comment that you are attempting to do it, and shortly your own personal label matching your GitHub ID will be assigned to that issue.

Feel free to propose issues that aren't described!

Tips

1. **starter** labeled issues are deemed to be good low-hanging fruit for newcomers to the project, Django, or even Python.
2. **doc** labeled issues must only touch content in the docs folder.

Setting up topic branches and generating pull requests

While it's handy to provide useful code snippets in an issue, it is better for you as a developer to submit pull requests. By submitting pull request your contribution to Django Packages will be recorded by Github.

In git it is best to isolate each topic or feature into a "topic branch". While individual commits allow you control over how small individual changes are made to the code, branches are a great way to group a set of commits all related to one feature together, or to isolate different efforts when you might be working on multiple topics at the same time.

While it takes some experience to get the right feel about how to break up commits, a topic branch should be limited in scope to a single `issue` as submitted to an issue tracker.

Also since GitHub pegs and syncs a pull request to a specific branch, it is the **ONLY** way that you can submit more than one fix at a time. If you submit a pull from your develop branch, you can't make any more commits to your develop without those getting added to the pull.

To create a topic branch, its easiest to use the convenient `-b` argument to `git checkout`:

```
git checkout -b fix-broken-thing
Switched to a new branch 'fix-broken-thing'
```

You should use a verbose enough name for your branch so it is clear what it is about. Now you can commit your changes and regularly merge in the upstream develop as described below.

When you are ready to generate a pull request, either for preliminary review, or for consideration of merging into the project you must first push your local topic branch back up to GitHub:

```
git push origin fix-broken-thing
```

Now when you go to your fork on GitHub, you will see this branch listed under the "Source" tab where it says "Switch Branches". Go ahead and select your topic branch from this list, and then click the "Pull request" button.

Here you can add a comment about your branch. If this in response to a submitted issue, it is good to put a link to that issue in this initial comment. The repo managers will be notified of your pull request and it will be reviewed (see below for best practices). Note that you can continue to add commits to your topic branch (and push them up to GitHub) either if you see something that needs changing, or in response to a reviewer's comments. If a reviewer asks for changes, you do not need to close the pull and reissue it after making changes. Just make the changes locally, push them to GitHub, then add a comment to the discussion section of the pull request.

Pull upstream changes into your fork regularly

Django Packages is advancing quickly. It is therefore critical that you pull upstream changes from develop into your fork on a regular basis. Nothing is worse than putting in a days of hard work into a pull request only to have it rejected because it has diverged too far from develop.

To pull in upstream changes:

```
git remote add upstream https://github.com/opencomparison/opencomparison.git
git fetch upstream develop
```

Check the log to be sure that you actually want the changes, before merging:

```
git log upstream/develop
```

Then merge the changes that you fetched:

```
git merge upstream/develop
```

For more info, see http://help.github.com/fork-a-repo/

How to get your pull request accepted

We want your submission. But we also want to provide a stable experience for our users and the community. Follow these rules and you should succeed without a problem!

Run the tests!

Before you submit a pull request, please run the entire Django Packages test suite via:

```
python manage.py test --settings=settings.test
```

The first thing the core committers will do is run this command. Any pull request that fails this test suite will be **rejected**.

If you add code/views you need to add tests!

We've learned the hard way that code without tests is undependable. If your pull request reduces our test coverage because it lacks tests then it will be **rejected**.

For now, we use the Django Test framework (based on unittest).

Also, keep your tests as simple as possible. Complex tests end up requiring their own tests. We would rather see duplicated assertions across test methods then cunning utility methods that magically determine which assertions are needed at a particular stage. Remember: *Explicit is better than implicit*.

Don't mix code changes with whitespace cleanup

If you change two lines of code and correct 200 lines of whitespace issues in a file the diff on that pull request is functionally unreadable and will be **rejected**. Whitespace cleanups need to be in their own pull request.

Keep your pull requests limited to a single issue

Django Packages pull requests should be as small/atomic as possible. Large, wide-sweeping changes in a pull request will be **rejected**, with comments to isolate the specific code in your pull request. Some examples:

1. If you are making spelling corrections in the docs, don't modify the settings.py file (pydanny is guilty of this mistake).
2. Adding new *Repo Handlers* must not touch the Package model or its methods.
3. If you are adding a new view don't '*cleanup*' unrelated views. That cleanup belongs in another pull request.
4. Changing permissions on a file should be in its own pull request with explicit reasons why.

Follow PEP-8 and keep your code simple!

Memorize the Zen of Python:

```
>>> python -c 'import this'
```

Please keep your code as clean and straightforward as possible. When we see more than one or two functions/methods starting with *_my_special_function* or things like *__builtins__.object = str* we start to get worried. Rather than try and figure out your brilliant work we'll just **reject** it and send along a request for simplification.

Furthermore, the pixel shortage is over. We want to see:

- *package* instead of *pkg*
- *grid* instead of *g*
- *my_function_that_does_things* instead of *mftdt*

Test any css/layout changes in multiple browsers

Any css/layout changes need to be tested in Chrome, Safari, Firefox, IE8, and IE9 across Mac, Linux, and Windows. If it fails on any of those browsers your pull request will be **rejected** with a note explaining which browsers are not working.

How pull requests are checked, tested, and done

First we pull the code into a local branch:

```
git checkout -b <branch-name> <submitter-github-name
git pull git://github.com/<submitter-github-name/django-twoscoops-project.git develop
```

Then we run the tests:

```
python manage.py test --settings=settings.test
```

We finish with a merge and push to GitHub:

```
git checkout develop
git merge <branch-name>
git push origin develop
```

Repo Handlers

This document describes the Django Packages Repo Handler API.

Adding a new repo system like Github in Django Packages is a relatively straightforward task. You need to provide two things:

1. Add a new repo handler in the apps.models.repos directory that follows the described API
2. Add tests to check your work
3. Document any special settings.
4. Change the SUPPORTED_REPO to include the name of the new repo handler.

What if my target repo doesn't support all the necessary fields?

Lets say you want to use *GitBlarg*, a new service whose API doesn't provide the number of repo_watchers or participants. In order to handle them you would just set those values until such a time as *GitBlarg* would support the right data.

For example, as you can see in the *apps.models.repos.base_handler.BaseHandler.fetch_metadata()* method, the Package instance that it expects to see is a comma-seperated value:

```
def fetch_metadata(self, package):
    """ Accepts a package.models.Package instance:

            return: package.models.Package instance

        Must set the following fields:

            package.repo_watchers (int)
            package.repo_forks (int)
            package.repo_description (text )
            package.participants = (comma-seperated value)

    """
    raise NotImplemented()
```

So your code might do the following:

```
from GitBlargLib import GitBlargAPI
def fetch_metadata(self, package):

    # fetch the GitBlarg data
    git_blarg_data = GitBlargAPI.get(package.repo_name())

    # set the package attributes
    package.repo_watchers = 0 # GitBlagAPI doesn't have this so we set to 0
    package.repo_forks = git_blarg_data.forks
    package.repo_description = git_blarg_data.note
    package.participants = u"" # GitBlagAPI doesn't have this so we set to an empty␣
↪string

    return package
```

How about cloning GitBlarg's repos so we can get a better view of the data?

The problem is that developers, designers, and managers will happily put gigabytes of data into a git/hg/svn/fossil/cvs repo. For a single project that doesn't sound like much, but when you are dealing with thousands of packages in a Django Packages instance the scale of the data becomes... well... terrifying. What is now a mild annoyance becomes a staggeringly large problem.

Therefore, pull requests on repo handlers that attempt to solve the problem this way will be summarily **rejected**.

Can I make a repo handler for Google Project Hosting?

Not at this time. Please read the FAQ.

Webservice APIv3

This is the APIv3 documentation for Django Packages. It is designed to be language and tool agnostic.

API Usage

This API is limited to read-only GET requests. Other HTTP methods will fail. Only JSON is provided.

API Reference

Representation Formats

Representation formats

- JSON.
- UTF-8.

Base URI

URI	Resource	Methods
<http-my-domain.com>/api/v3/	Root	GET

URIs

URI	Resource	Methods
/	Index	GET
/*categories*/	Category list	GET
/*categories*/{slug}/	Category	GET
/*grids*/	Grid list	GET
/*grids*/{slug}/	Grid	GET
/**grid_**/{slug}/*packages*/	Grid Packages list	GET
/*packages*/	Package list	GET
/*packages*/{slug}/	Package	GET
/**users_**/{slug}/	User	GET

Resources

Categories

Representation:

```
{
    "absolute_url": "/categories/apps/",
    "show_pypi": true,
    "slug": "apps",
    "title_plural": "Apps",
    "created": "2010-08-14T22:47:52",
    "description": "Small components used to build projects. An app is anything that␣
↪is installed by placing in settings.INSTALLED_APPS.",
    "title": "App",
    "resource_uri": "/api/v3/categories/apps/",
    "modified": "2010-09-12T22:42:58.053"
}
```

Grids

Representation:

```
{
    absolute_url: "/grids/g/cms/",
    created: "Sat, 14 Aug 2010 20:12:46 -0400",
    description: "This is a list of Content Management System applications for Django.
↪",
    is_locked: false,
    modified: "Sat, 11 Sep 2010 14:57:16 -0400",
    packages: [
        "/api/v3/package/django-cms/",
        "/api/v3/package/mezzanine/",
        "/api/v3/package/django-page-cms/",
```

```
        "/api/v3/package/django-lfc/",
        "/api/v3/package/merengue/",
        "/api/v3/package/philo/",
        "/api/v3/package/pylucid/",
        "/api/v3/package/django-gitcms/",
        "/api/v3/package/django-simplepages/",
        "/api/v3/package/djpcms/",
        "/api/v3/package/feincms/",
    ],
    resource_uri: "/api/v3/grid/cms/",
    slug: "cms",
    title: "CMS"
}
```


Packages

Representation:


```
{
    "last_fetched": "2015-02-28T12:04:58.537",
    "slug": "django",
    "resource_uri": "/api/v3/packages/django/",
    "pypi_url": "http://pypi.python.org/pypi/Django",
    "repo_url": "https://github.com/django/django",
    "absolute_url": "/packages/p/django/",
    "commits_over_52": "67,38,76,55,35,34,52,52,35,42,63,61,46,61,70,65,43,48,34,24,
↪57,56,44,58,54,57,51,54,36,48,28,45,38,44,53,30,69,91,66,65,36,45,68,54,64,111,50,
↪36,60,31,0,0",
    "category": "/api/v3/categories/frameworks/",
    "created_by": null,
    "created": "2010-08-14T22:50:35",
    "repo_description": "The Web framework for perfectionists with deadlines.",
    "commit_list": "[78, 36, 42, 71, 62, 48, 41, 59, 48, 47, 33, 53, 33, 23, 28, 36,␣
↪45, 34, 36, 25, 38, 52, 45, 43, 111, 115, 58, 49, 52, 62, 50, 29, 25, 14, 20, 55,␣
↪97, 109, 60, 32, 38, 47, 60, 53, 49, 26, 43, 48, 55, 29, 73, 0]",
    "repo_watchers": 13087,
    "last_modified_by": null,
    "title": "Django",
    "grids": [
        "/api/v3/grids/file-streaming/",
        "/api/v3/grids/this-site/"
    ],
    "repo_forks": 5113,
    "pypi_version": "1.8b1",
    "documentation_url": "https://djangoproject.com",
    "participants": "adrianholovaty,malcolmt,freakboy3742,timgraham,aaugustin,claudep,
↪jezdez,jacobian,spookylukey,alex,ramiro,andrewgodwin,gdub,akaariai,kmtracey,jbronn,
↪pydanny,audreyr,etc",
    "modified": "2015-03-01T08:00:39.708",
    "usage_count": 356
}
```


User

Representation:

```
{
    "username": "jezdez",
    "last_login": "2014-09-21T07:37:17.619",
    "date_joined": "2010-08-21T07:14:03",
    "created": "2011-09-09T17:10:29.509",
    "absolute_url": "/profiles/jezdez/",
    "google_code_url": null,
    "github_account": "jezdez",
    "bitbucket_url": "jezdez",
    "modified": "2014-09-21T07:37:17.598",
    "resource_uri": "/api/v3/users/jezdez/"
}
```


Lessons Learned

Some of these are common sense, and others we learned during the events in question.

DjangoCon 2010

- For sprints, show up early the first day.
- Stay in a hotel near the sprint. If you have to spend an hour going each way that's up to 20% of sprint time you are wasting each day. If necessary, switch hotels.

PyCon 2011

Getting Sprinters

- Mark easy stuff for beginners. After they knock out an issue or two the stuff they've learned lets them handle harder tasks.
- Sit-down with each new contributor individually for at least 15 minutes to help them through the installation process. They get started much faster. you'll spot the mistakes in your docs, and they'll hang around longer.
- If you see anyone during the sprints who looks lost or without a project, invite them to join you.
- If you have a full sprint table and a non-sprinter is sitting with you get them to contribute something small. They go from being a distraction to a valued member of the team.
- Go out for dinner at a fun restaurant the first night with just your team. On other nights try to keep meals short since long meals mean hours of missed sprint time.

Assigning Work

- Assign issues in the issue tracker to specific people. No one should work a task unless they have had it assigned to them. This way you avoid duplication of effort.
- Tell people if they get stuck on something for 30 minutes to ask questions. We are all beginners and the hardest problems often become simple spelling mistakes when you try and explain them.

Be conservative

You don't want to stall people from doing the work they are trying to get done. So that means:

- Keep the database as stable as possible during a large sprint.
- Freeze the design during a sprint. Have designer-oriented people prettify neglected views e.g. the login page, server error pages.

Helping people get stuff done

- If you are leading a sprint don't expect to get any code done yourself. Your job is to facilitate other people to have fun hacking, learning, and getting things done.
- Go around and ask questions of your sprinters periodically. People are often too shy to come up to you but if you go up to them they'll readily ask for help.
- Update your install documentation as your sprinters discover problems.
- If you have new dependencies, let everyone know as soon and as loudly as possible.
- Good documentation is as important as code. When people ask questions rather than just answering the question, walk them through the specific answer in your docs. If the answer doesn't exist, document it yourself and have them help you write the answer.
- Demonstrate coverage.py to the sprinters, show them how to write tests, and provide good test examples. Good test coverage will save everyone a lot of grief during development and deployment.
- Have your code working on all major platforms with installation instructions for each platform. Your code on all platforms will be that much stable for it.
- Have a portable drive with the dependencies for your project on it. You can never count on the network being reliable at a sprint.
- If a beginning developer asks for help, try to get your advanced sprinters to answer the questions and possibly pair with them for a while.
- When someone is working really hard and is trying to focus, run interference for them.

Pull Requests

- Provide good and bad pull request examples.
- Don't be afraid of sounding stupid if you don't understand someone's pull request. If it confuses you it's going to confuse newcomers even more and hence make your code unmaintainable. Remember that simplicity is a virtue and is one of the best things of projects like Python, Pyramid, and Flask.
- Each time someone submits a pull request, ask them if they've run the full test suite. Yeah, it's repetitive but they'll thank you for it.
- If someone submits a broken pull request, see if you can work out the issue with them. If the problem is not easily corrected, ask them to fix the problem and resubmit the pull request.

CHAPTER 2

In development

- caching

PostgreSQL setup instructions for new contributors

Mac

EnterpriseDB maintains a Mac OS X binary installer. First, download and install from here:

http://www.enterprisedb.com/products-services-training/pgdownload#osx

The package will take care of most of the PostgreSQL installation needs but it needs a couple of small tweaks.

Become the new postgres user that the package added:

```
sudo su - postgres
```

Source the environment file:

```
source pg_env.sh
```

Next, setup postgres to listen on TCP/IP sockets. Edit `$PGDATA/postgresql.conf` and listen_addresses is set to 'localhost'.

Also, for a more convenient development server setup, it is nice to loosen the host-based security settings for localhost. Edit `$PGDATA/pg_hba.conf` and set the local and 127.0.0.1/32 lines to use "trust" authentication (change the last column from md5 to trust).

Lastly, apply the changes using `pg_ctl reload` and `exit` to log out as the postgres user.

Now you should be able to access postgres using `psql -U postgres`. Create a new database using `createdb -U postgres opencomparison`.

Another way

If you prefer to use Homebrew to install your software you can do this:

```
brew install postgresql
initdb /usr/local/var/postgres -E utf8
pg_ctl -D /usr/local/var/postgres -l /usr/local/var/postgres/server.log start
```

Change the path used in `initdb` and other commands if you'd rather store your data files somewhere other than `/usr/local/var/postgres`.

Once the server is started, execute:

```
createdb opencomparison
```

Then you should be able to access the database you created via `psql` so:

```
psql --dbname opencomparison
```

Remeber to shut down the service when not in use:

```
pg_ctl -D /usr/local/var/postgres stop
```

The security defaults are already in place, and will allow a lot of access. This should never be considered a production-ready deployment scenario.

Ubuntu

Install Postgres 8.4 (the version used on the site, as of this writing) with:

```
sudo apt-get install postgresql-8.4 libpq-dev
```

Edit `/etc/postgresql/8.4/main/postgresql.conf` and make sure the listen line is either `listen = 'localhost'` or `listen = '*'` to listen on all interfaces.

Also, for a more convenient development server setup, it is nice to loosen the host-based security settings for localhost. Edit `/etc/postgresql/8.4/main/pg_hba.conf` and set the local and 127.0.0.1/32 lines to use "trust" authentication (change the last column from md5 to trust).

Apply those changes with `/etc/init.d/postgresql-8.4 reload`.

Lastly, create a new database using `createdb -U postgres opencomparison`.

Windows

EnterpriseDB maintains a Windows binary installer. First, download and install from here:

http://www.enterprisedb.com/products-services-training/pgdownload#windows

The package will take care of most of the PostgreSQL installation needs but it needs a couple of small tweaks.

Install the Windows port of psycopg2 from http://www.stickpeople.com/projects/python/win-psycopg/

Open pgAdmin III. Right-click on PostgreSQL 8.4 (localhost:5432) and choose Connect. Enter the Postgres user password.

Right-click Databases and choose New Database. Give it the name opencomparison and the owner postgres. Click OK.

CHAPTER 3

Credits

For Django Dash 2010, @pydanny and @audreyr were scared of rabbits.

Since then the project has had many contributors.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search