OpenComparison Documentation

Release 0.9.0 beta

Audrey Roy, Daniel Greenfeld and contributors

October 25, 2012

CONTENTS

OpenComparison 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? OpenComparison solves that problem for you!

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

CHAPTER

ONE

CONTRIBUTING TO OPENCOMPARISON

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

Contents:

1.1 Introduction

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

Packaginator stores information on fetched packages and provides easy comparison tools for them. Public APIs include PyPI, Github, BitBucket, Launchpad, and perhaps soon SourceForge and Google Project Hosting.

1.1.1 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

- Launchpad.

Google Project Hosting and Sourceforge are not fully supported!

Not yet!

The progenitor of Packaginator, Django Packages was cooked up during Django Dash 2010. We wanted to keep the scope of our work reasonable. We'll try to include more sites in the future. Here are some details:

- Sourceforge needs needs to repair their API and then we can play.
- Google's lack of a formal API leaves us the option of screen-scraping their content. We're not excited about introducing that sort of brittle activity into Packaginater.

1.2 License

1.3 Installation

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

1.3.1 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
```

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)

No. See the faq.

1.3.2 Main instructions

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

Git clone the project and install requirements

Create a virtualenv, activate it, git clone the OpenComparison 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/mkii.txt
```

Set up local settings

In the `settings/` directory, Copy the `local_settings.py.example` to `local_settings.py`:

```
cd settings
cp local_settings.py.example local_settings.py
```

Change the `ROOT_URLS` setting in `local_settings.py` from *<root_directory_name>* to the correct value (i.e. the name of your repo):

```
ROOT_URLCONF = '<root_directory_name>.urls'
```

OPTIONAL! You can enable launchpad support in the local settings file. Launchpad's dependencies can be a little fussy, so this will probably require some additional tweaking on your part:

```
LAUNCHPAD_ACTIVE = False
```

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] '
```

Change the `SECRET_KEY` setting in `local_settings.py` to your own secret key:

```
SECRET_KEY = "CHANGE-THIS-KEY-TO-SOMETHING-ELSE"
```

Set up your PostgreSQL database

Set up PostgreSQL and create a database as per the postgresql contributor instructions.

Make your database:

```
python manage.py syncdb
python manage.py migrate
```

OPTIONAL! Load some base data for development usage. This should not be loaded on the production site:

```
python manage.py loaddata
```

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
```

1.4 Optional Celery install & config

Install redis (Mac):

```
brew install redis
```

Install redis (Ubuntu):

```
apt-get redis
```

Necessary changes to local_settings.py:

```
CELERY_ALWAYS_EAGER = False
CELERY_EAGER_PROPAGATES_EXCEPTIONS = True
BROKER_BACKEND = "redis"
BROKER_HOST = "localhost"
BROKER_POST = 6379
BROKER_VHOST = "0"
```

Start redis:

```
redis-server
```

In the Django shell enter to fire off queue:

```
python manage.py shell
>>> from pypi.tasks import queue_all_pypi_packages
>>> queue_all_pypi_packages()
```

In another Django shell:

```
python manage.py celeryd -c 4
```

1.5 Troubleshooting

1.5.1 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
```

1.5.2 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
```

1.5.3 ImportError related to launchpad.py

Sometimes this shows up as "Caught ImportError while rendering: cannot import name ScalarValue".

You're having Launchpad/bzr installation problems. Most likely cause is your C compiler. On Windows, make sure you have MinGW installed as per the installation instructions. On Linux, make sure you have the python-dev and gcc packages.

1.5.4 I can't get it to work in buildout!

We don't support buildout. See the faq.

1.5.5 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)

1.5.6 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

1.6 FAQ

1.6.1 General

How did Open Comparison 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

Is there an on-line community?

We'll be opening an IRC channel shortly.

How can I contribute?

Read the page on contributions.

What browsers does Open Comparison support?

We do formal tests on Chrome, Safari, Firefox, IE8, and IE9.

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.

1.6.2 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. 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 Open Comparison, 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
```

1.6.3 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?

Open Comparison 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.

1.7 Settings

How to customize the settings to suit your needs. Do this in local_settings so patches and upstream pulls don't cause havoc to your installation

1.7.1 FRAMEWORK_TITLE (Default: "Django")

Used to create the name of the site.

1.7.2 PACKAGE_EXTENDERS

Used to determine how packages have extended data sets. See package_extenders

1.7.3 PACKAGINATOR_SEARCH_PREFIX (Default: "django")

In the case of **Django Packages**, autocomplete searches for something like 'forms' was problematic because so many packages start with 'django'. The same will hold for searches in **Python Packages** and **Pyramid Packages**. This prefix is accommodated in searches to prevent this sort of problem.

example:

```
PACKAGINATOR_SEARCH_PREFIX = 'pyramid'
```

1.7.4 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/uberi
    "PYPI_URL" : "<strong>Leave this blank if this package does not have a PyPI release.</strong><br
}
```

1.7.5 Launchpad Specific settings

The launchpad Python client tool requires an unbelievable amount of requirements to handle a simple JSON ReST based webservice. These requirements can be tricky to install. Therefore, Packaginator out of the box does not support Launchpad.

If you have problems, please refer to troubleshooting.

LAUNCHPAD_ACTIVE (Default: False)

If you want your instance of Packaginator to support Launchpad, set this setting to true in local_settings.py:

```
LAUNCHPAD_ACTIVE = True
```

LAUNCHPAD_CACHE_DIR

Used to point LAUNCHPAD commands against the appropriate cache. Important in real hosting machines.

Example:

```
LAUNCHPAD_CACHE_DIR = "/tmp/lp-cache"
```

1.7.6 Permissions Settings

Packaginator provides several ways to control who can make what changes to things like packages, features, and grids. By default, a Packaginator 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

1.8 Testing Instructions

1.8.1 Running the test suite

To run all of the Packaginator tests:

```
python manage.py test
```

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

```
python manage.py test feeds
```

1.8.2 Testserver

Did you know that Django has a built-in testserver that lets you quickly run a development server with data from any fixture?

To run the test server with a particular Packaginator fixture, for example with test_initial_data.json:

```
python manage.py testserver test_initial_data
```

Open up a web browser. You'll see the Packaginator site, populated with test data from that file.

1.9 Management Commands

1.9.1 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.

1.10 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
```

1.11 Team

1.11.1 Project Leads

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

1.11.2 Core Developers at DjangoCon 2011

- James Punteney
- Mike Johnson
- Taylor Mitchell

1.11.3 Core Developers at DataMigrationCon 2011

- Katharine Jarmul

1.11.4 Core Developers at PyCon 2011

- Gisle Aas
- Nate Aune
- Szilveszter Farkas

1.11.5 Core Developers at DjangoCon 2011

- James Punteney
- Jonas Obrist
- Taavi Taijala

1.11.6 Direct Contributors

- Aaron Kavlie
- Adam Saegebarth
- Alex Robbins
- Andrii Kurinny
- Brian Ball
- Bryan Weingarten
- Chris Adams
- David Peters
- Eric Spunagle
- Evgeny Fadeev
- Flaviu Simihaian
- Gisle Aas (Repo Man)
- Jacob Burch
- James Pacileo
- James Punteney
- Jeff Schenck
- Jim Allman
- John M. Camara
- Jonas Obrist
- jrothenbuhler
- Kenneth Love
- Mike Johnson
- Nate Aune
- Nolan Brubaker
- PA Parent
- Preston Holmes
- Skot Carruth
- Stuart Powers
- Szilveszter Farkas (Repo Man)

- Taavi Taijala
- Taylor Mitchell
- Tom Brander
- Vasja Volin

1.11.7 Other Contributors

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

1.12 Contributing

1.12.1 Setup

Fork on GitHub

Before you do anything else, login/signup on GitHub and fork OpenComparison 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 OpenComparison

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

1.12.2 Issues!

The list of outstanding OpenComparison 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.

1.12.3 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 OpenComparison 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 master branch, you can't make any more commits to your master 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 master 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.

1.12.4 Pull upstream changes into your fork regularly

OpenComparison is advancing quickly. It is therefore critical that you pull upstream changes from master 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 master.

To pull in upstream changes:

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

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

```
git log upstream/master
```

Then merge the changes that you fetched:

```
git merge upstream/master
```

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

1.12.5 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 OpenComparison test suite via:

```
python manage.py 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

OpenComparison 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 a new repo handler 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.

1.12.6 How pull requests are checked, tested, and done

First we pull the code into a local branch:

```
git remote add <submitter-github-name> git@github.com:<submitter-github-name>/opencomparison.git
git fetch <submitter-github-name>
git checkout -b <branch-name> <submitter-github-name>/<branch-name>
```

Then we run the tests:

```
python manage.py test
```

We finish with a non-fastforward merge (to preserve the branch history) and push to GitHub:

```
git checkout master
git merge --no-ff <branch-name>
git push upstream master
```

1.13 Repo Handler API

Adding a new repo in Packaginator 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.

1.13.1 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
```

1.13.2 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 Packaginator 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**.

1.13.3 Can I make a repo handler for Google Project Hosting?

Not at this time. Please read the FAQ.

1.14 Webservice API

This is the API documentation for OpenComparison. It is designed to be language and tool agnostic.

1.14.1 API Usage

Only JSON is provided

1.14.2 API Reference

Representation Formats

Representation formats

- JSON.
- UTF-8.

Base URI

URI	Resource	Methods
<http-my-domain.com>/api/v1/	Root	GET

URIs

URI	Resource	Methods
/category/	Category list	GET, POST
/category/{slug}/	Category	GET, POST
/grid/	Grid list	GET, POST
/grid/{slug}/	Grid	GET, POST
/grid/{slug}‘/packages‘_/	Grid Packages list	GET, POST
/grid-of-the-week/	Featured Grid list	GET, POST
/grid-of-the-week/{slug}/	Featured Grid	GET, POST
/package/	Package list	GET, POST
/package/{slug}/	Package	GET, POST
/package-of-the-week/	Featured Package list	GET, POST
/package-of-the-week/{slug}/	Featured Package	GET, POST

Resources

Category

Representation:

```
{
    created: "Sat, 14 Aug 2010 19:47:52 -0400"
    description: "Small components used to build projects. An app is anything that is installed by pl
    modified: "Sat, 28 Aug 2010 11:20:36 -0400"
    resource_uri: "/api/v1/category/apps/"
    slug: "apps"
    title: "App"
    title_plural: "Apps"
}
```

Grid

Representation:

```
{
    absolute_url: "/grids/g/cms/"
    created: "Sat, 14 Aug 2010 20:12:46 -0400"
    description: "This page lists a few well-known reusable Content Management System applications fo
    is_locked: false
    modified: "Sat, 11 Sep 2010 14:57:16 -0400"
    packages: [
        "/api/v1/package/django-cms/"
        "/api/v1/package/django-page-cms/"
        "/api/v1/package/django-lfc/"
        "/api/v1/package/merengue/"
        "/api/v1/package/mezzanine/"
        "/api/v1/package/philo/"
        "/api/v1/package/pylucid/"
        "/api/v1/package/django-gitcms/"
        "/api/v1/package/django-simplepages/"
        "/api/v1/package/djpcms/"
        "/api/v1/package/feincms/"
    ]
    resource_uri: "/api/v1/grid/cms/"
    slug: "cms"
    title: "CMS"
}
```


Grid-of-the-week

Representation:


```
{
    absolute_url: "/grids/g/cms/"
    created: "Sun, 15 Aug 2010 01:36:59 -0400"
    end_date: "22 Aug 2010"
    modified: "Sun, 15 Aug 2010 01:36:59 -0400"
    resource_uri: "/api/v1/grid-of-the-week/cms/"
    start_date: "15 Aug 2010"
}
```


Package

Representation:


```
{
    absolute_url: "/packages/p/pinax/"
    category: "/api/v1/category/frameworks/"
    created: "Mon, 16 Aug 2010 23:25:16 -0400"
    grids: [
        "/api/v1/grid/profiles/"
        "/api/v1/grid/social/"
        "/api/v1/grid/this-site/"
    ]
    modified: "Sun, 12 Sep 2010 17:02:10 -0400"
    participants: "pinax,brosner,jtauber,jezdez,ericflo,gregnewman,pydanny,edcrypt,paltman,dougn,alex
    pypi_downloads: 0
    pypi_url: "http://pypi.python.org/pypi/Pinax"
    pypi_version: "0.9a1"
```

```
    repo: "/api/v1/repo/1/"
    repo_commits: 0
    repo_description: "a Django-based platform for rapidly developing websites"
    repo_forks: 184
    repo_url: "http://github.com/pinax/pinax"
    repo_watchers: 913
    resource_uri: "/api/v1/package/pinax/"
    slug: "pinax"
    title: "Pinax"
}
```

Package-of-the-week

Representation:

```
{
    absolute_url: "/packages/p/django-uni-form/"
    created: "Sun, 15 Aug 2010 01:36:38 -0400"
    end_date: "15 Aug 2010"
    modified: "Mon, 16 Aug 2010 23:54:36 -0400"
    resource_uri: "/api/v1/package-of-the-week/django-uni-form/"
    start_date: "14 Aug 2010"
}
```

1.15 Lessons Learned

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

1.15.1 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.

1.15.2 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.

1.16 Development reference documentation

1.16.1 Top level module `apps`

Module containing the django applications that together make up the packaginator site.

All of the modules listed below are the sub-applications.

1.16.2 `apiv1` - restful API

Restful api for the packaginator, based on `django-tastypie` (docs | pypi | repo).

This module consists of two components - module `api` and the resource definition module `resources`.

The api urls are exposed in project's main `urls.py` file

`apiv1.api`

The `Api` definition module

class `apps.apiv1.api.`**`Api`**(*api_name='v1'*)
A sub-class of `TastyPieApi` - the actual Api class

`top_level`(*request*, *api_name=None*)
A view that returns a serialized list of all resources registered to the `Api`. Useful for the resource discovery.

`apiv1.resources`

1.16.3 `core` - Kernal bits

`core.fields`

class `apps.core.fields.`**`CreationDateTimeField`**(**args*, ***kwargs*)

`south_field_triple`()
Returns a suitable description of this field for South.

class `apps.core.fields.`**`ModificationDateTimeField`**(**args*, ***kwargs*)

`south_field_triple`()
Returns a suitable description of this field for South.

`core.models`

class `apps.core.models.`**`BaseModel`**(**args*, ***kwargs*)
Base abstract base class to give creation and modified times

1.16.4 feeds - RSS and Atom feeds

This application defines RSS and Atom feeds that are made available to the users of the packaginator

`feeds.urls` url patterns for the feeds

`feeds.feeds` Contains classes for the feeds

class `apps.feeds.feeds.`**AtomLatestPackagesFeed**
Atom feed for the packages

class `apps.feeds.feeds.`**RssLatestPackagesFeed**
RSS Feed for the packages

item_description(*item*)
Get description of the repository

item_pubdate(*item*)
Get publication date

item_title(*item*)
Get title of the repository

items()
Returns 15 most recently created repositories

1.16.5 grid - package grid app

Grid application - displays and manipulates the package grid

grid.views

views for the `apps.grid` app

`grid.views.`**add_feature**(*request*, **args*, ***kwargs*)
Adds a feature to the grid, accepts GET and POST requests.

Requires user to be logged in

Template context:

- `form` - instance of `grid.forms.FeatureForm` form
- `grid` - instance of `grid.models.Grid` model

`grid.views.`**add_grid**(*request*, **args*, ***kwargs*)
Creates a new grid, requires user to be logged in. Works for both GET and POST request methods

Template context:

- `form` - an instance of `GridForm`

`grid.views.`**add_grid_package**(*request*, **args*, ***kwargs*)
Add an existing package to this grid.

`grid.views.`**add_new_grid_package**(*request*, **args*, ***kwargs*)
Add a package to a grid that isn't yet represented on the site.

`grid.views.`**ajax_grid_list**(*request*, *template_name='grid/ajax_grid_list.html'*)

`grid.views.`**build_element_map**(*elements*)

`grid.views.`**delete_feature**(*request*, **args*, ***kwargs*)

grid.views.**delete_grid**(*request*, **args*, ***kwargs*)
Deletes a grid, requires user to be logged in.

grid.views.**delete_grid_package**(*request*, *id*, *template_name='grid/edit_feature.html'*)
Deletes package from the grid, `id` is the id of the `grid.models.GridPackage` instance

Requires permission `grid.delete_gridpackage`.

Redirects to `grid.views.grid_detail()`.

grid.views.**edit_element**(*request*, **args*, ***kwargs*)

grid.views.**edit_feature**(*request*, **args*, ***kwargs*)
edits feature on a grid - this view has the same semantics as `grid.views.add_feature()`.

Requires the user to be logged in.

grid.views.**edit_grid**(*request*, **args*, ***kwargs*)
View to modify the grid, handles GET and POST requests. This view requires user to be logged in.

Template context:

- `form` - instance of `grid.forms.GridForm`

grid.views.**grid_detail**(*request*, *slug*, *template_name='grid/grid_detail.html'*)
displays a grid in detail

Template context:

- `grid` - the grid object
- `elements` - elements of the grid
- `features` - feature set used in the grid
- `grid_packages` - packages involved in the current grid

grid.views.**grid_detail_feature**(*request*, *slug*, *feature_id*, *bogus_slug*, *template_name='grid/grid_detail_feature.html'*)
a slightly more focused view than `grid.views.grid_detail()` shows comparison for only one feature, and does not show the basic grid parameters

Template context is the same as in `grid.views.grid_detail()`

grid.views.**grids**(*request*, *template_name='grid/grids.html'*)
lists grids

Template context:

- `grids` - all grid objects

grid.models

class grid.models.**Element**(**args*, ***kwargs*)
The individual cells on the grid. The `Element` grid attributes are:

- `grid_package` - foreign key to `GridPackage`
- `feature` - foreign key to `Feature`
- `text` - the actual contents of the grid cell

class grid.models.**Feature**(**args*, ***kwargs*)
These are the features measured against a grid. `Feature` has the following attributes:

- `grid` - the grid to which the feature is assigned

- `title` - name of the feature (100 chars is max)
- `description` - plain-text description

class `grid.models.`**`Grid`**(**args*, ***kwargs*)
Grid object, inherits form `package.models.BaseModel`. Attributes:

- `title` - grid title
- `slug` - grid slug for SEO
- `description` - description of the grid with line breaks and urlized links
- `is_locked` - boolean field accessible to moderators
- `packages` - many-to-many relation with :class:~'grid.models.GridPackage' objects

`grid_packages`
Gets all the packages and orders them for views and other things

class `grid.models.`**`GridPackage`**(**args*, ***kwargs*)
Grid package. This model describes packages listed on one side of the grids and explicitly defines the many-to-many relationship between grids and the packages (i.e - allows any given package to be assigned to several grids at once).

Attributes:

- `grid` - the `Grid` to which the package is assigned
- `package` - the `Package`

`grid.forms`

Forms for the `grid` app

class `grid.forms.`**`ElementForm`**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)
collects data for a new grid element - a `ModelForm` for `grid.models.Element`

class `grid.forms.`**`FeatureForm`**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)
collects data for the feature - a `ModelForm` for `grid.models.Feature`

class `grid.forms.`**`GridForm`**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)
collects data for the new grid - a django `ModelForm` for `grid.models.Grid`

`clean_slug`()
returns lower-cased slug

class `grid.forms.`**`GridPackageForm`**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)
collects data for a new package - a `ModelForm` for `grid.models.GridPackage`

1.16.6 `profiles` - profiles app

Manages user profiles

profiles.models

class profiles.models.**Profile**(**args*, ***kwargs*)
Profile(id, created, modified, user_id, github_account, github_url, bitbucket_url, google_code_url, email)

my_packages()
Return a list of all packages the user contributes to.

List is sorted by package name.

save(***kwargs*)
Override save to always populate email changes to auth.user model

url_for_repo(*repo*)
Return the profile's URL for a given repo.

If url doesn't exist return None.

profiles.context_processors

profiles.context_processors.**lazy_profile**(*request*)
Returns context variables required by templates that assume a profile on each request

1.16.7 pypi - pypi app

All connection points with PyPI

pypi.slurper

PyPI interface (see http://wiki.python.org/moin/PyPiXmlRpc)

class pypi.slurper.**Slurper**(*package*)
Fetches data from PyPI

get_latest_version_number(*package_name*, *versions=None*)
Returns the latest version number for a package

get_or_create_package(*package_name*, *version*)

pypi.versioning

pypi.versioning.**compare_versions**(*version1*, *version2*)
Determines the order of versions

pypi.versioning.**highest_version**(*versions*)
returns the highest version

1.16.8 searchv2 - searchv2 app

searchv2.views

searchv2.views.**build_search**(*request*, **args*, ***kwargs*)

searchv2.views.**search**(*request*, *template_name='searchv2/search.html'*)
Searches in Grids and Packages

searchv2.views.**search_function**(*q*)
TODO - make generic title searches have lower weight

searchv2.views.**search_packages_autocomplete**(*request*)
Searches in Packages

CHAPTER

TWO

IN DEVELOPMENT

- caching

2.1 Package Extenders

Warning: This is a work in progress. Much of the work was done in the package_refactor branch.

2.1.1 What remains

- Get apiv2 in place since apiv1 is broken hard

2.1.2 How it works

```
settings.PACKAGE_EXTENDERS = [
    {
        'form':'apps.dummy.forms.DummyForm',
        'model':'dummy.DummyModel'
        # form
        # model
        # grid_items
        # package_displays
    },
]
```

Originally OpenComparison packages just dealt with packages stored in the Python Package Index (PyPI) and with extra data provided by common repo systems like Bitbucket, Github and Launchpad. The purpose of this setting is to remove the tight coupling used for that and allow for Packages. This abstraction is designed to allow Python apps that follow a standard interface to be plugged seamlessly into OpenComparison, and unplugged - all without additional wiring in regards to settings, templates, and urls.

The interface system is described as follows:

forms

- Can provide extra fields for use in the add/edit PackageForm
- Can provide add/edit forms for use on Package detail page to capture extra data. e.g. examples.

models (optional)

Models can be assigned to related to package.models.Package:

```
class DummyModel(object):

    package = models.ForeignKey(Package)
```

templates

- TODO provide style guide
- TODO allow for inlines/blocks that can be looped through if named correctly. Sample
  - TODO xyz/snippets/_grid.html
  - TODO xyz/snippets/_package.html

urls

- standard
- Added via loop in root urls.py on settings.PACKAGE_EXTENDERS

views

- standard
- Added thanks to urls.py

CHAPTER

THREE

CREDITS

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

Since then the project has had many contributors.

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

a

apps, **??**
apps.apiv1, **??**
apps.apiv1.api, **??**
apps.core, **??**
apps.core.fields, **??**
apps.core.models, **??**
apps.feeds, **??**
apps.feeds.feeds, **??**
apps.feeds.urls, **??**
apps.grid, **??**
apps.profiles, **??**
apps.pypi, **??**
apps.searchv2, **??**

g

grid.forms, **??**
grid.models, **??**
grid.views, **??**

p

profiles.context_processors, **??**
profiles.models, **??**
pypi.slurper, **??**
pypi.versioning, **??**

s

searchv2.views, **??**