puput Documentation

Release 0.8

Marc Tudurí

Apr 30, 2017

Contents

Puput is a powerful and simple Django app to manage a blog. It uses the awesome Wagtail CMS as content management system.

Puput is the catalan name for Hoopoe which is indeed a beautiful bird.

CHAPTER 1

Features

- Built with Wagtail CMS and Django
- Inspired in Wordpress and Zinnia
- Simple & responsive HTML template by default
- SEO friendly urls
- Support for Disqus comments
- Entries by author, tags, categories, archives and search term
- Last & popular entries
- Configurable sidebar widgets
- RSS feeds
- Related entries
- Extensible entry model

CHAPTER 2

Contents:

Setup

If you're starting from a Django project without Wagtail integration and you want to add a blog site to your project, please follow the steps outlined under *Standalone blog app*. If you are already using Wagtail, refer to *Installation on top of Wagtail*.

Standalone blog app

1. Install Puput and its dependencies via `pip install puput`.
2. Append `PUPUT_APPS` to `INSTALLED_APPS` in your settings.

```
from puput import PUPUT_APPS

INSTALLED_APPS += PUPUT_APPS
```

This includes Puput, Wagtail's apps and certain third-party dependencies. If you are already referencing one of these apps in your `INSTALLED_APPS` list, please include the following apps manually in order to avoid app collisions:

```
INSTALLED_APPS = (
    ...
    'wagtail.wagtailcore',
    'wagtail.wagtailadmin',
    'wagtail.wagtaildocs',
    'wagtail.wagtailsnippets',
    'wagtail.wagtailusers',
    'wagtail.wagtailimages',
    'wagtail.wagtailembeds',
    'wagtail.wagtailsearch',
    'wagtail.wagtailsites',
    'wagtail.wagtailredirects',
    'wagtail.wagtailforms',
```

```
    'wagtail.contrib.wagtailsitemaps',
    'wagtail.contrib.wagtailroutablepage',
    'compressor',
    'taggit',
    'modelcluster',
    'puput',
)
```

3. Add Wagtail's required middleware classes to `MIDDLEWARE_CLASSES` in your Django settings.

```
MIDDLEWARE_CLASSES = (
    ...
    'wagtail.wagtailcore.middleware.SiteMiddleware',
    'wagtail.wagtailredirects.middleware.RedirectMiddleware',
)
```

4. Add the `request` context processor to the `TEMPLATE_CONTEXT_PROCESSORS` structure in your Django settings.

```
TEMPLATE_CONTEXT_PROCESSORS = (
    ...
    'django.core.context_processors.request',
)
```

5. Set the `WAGTAIL_SITE_NAME` variable to the name of your site in your Django settings.

```
WAGTAIL_SITE_NAME = 'Puput blog'
```

6. Configure the `MEDIA_ROOT` and `MEDIA_URL` settings as described in the Wagtail Docs.

```
MEDIA_ROOT = os.path.join(PROJECT_ROOT, 'media')
MEDIA_URL = '/media/'
```

7. Place Puput's URLs at the **bottom** of the urlpatterns. It also includes Wagtail's URLs.

```
urlpatterns = [
    ...
    url(r'', include('puput.urls')),
]
```

8. To make your Django project serve your media files (e.g. things you upload via the admin) during development, don't forget to add this to your urlpatterns:

```
from django.conf import settings

if settings.DEBUG:
    import os
    from django.conf.urls.static import static
    from django.views.generic.base import RedirectView
    from django.contrib.staticfiles.urls import staticfiles_urlpatterns

    urlpatterns += staticfiles_urlpatterns() # tell gunicorn where static files are␣
↪in dev mode
    urlpatterns += static(settings.MEDIA_URL + 'images/', document_root=os.path.
↪join(settings.MEDIA_ROOT, 'images'))
    urlpatterns += [
```

```
        (r'^favicon\.ico$', RedirectView.as_view(url=settings.STATIC_URL + 'myapp/
↪images/favicon.ico')),
    ]
```

9. Run `python manage.py migrate` and `python manage.py puput_initial_data` to load initial data to start a blog site.
10. Open your browser at http://127.0.0.1:8000/blog/ to view your blog home page. Go to http://127.0.0.1:8000/blog_admin/ to view the admin site and edit your content.

Installation on top of Wagtail

1. Install Puput and its dependencies via `pip install puput`.
2. Add `puput`, `compressor`, `wagtail.contrib.wagtailsitemaps` and `wagtail.contrib.wagtailroutablepage` to `INSTALLED_APPS` in your Django settings.
3. If you have previously defined Wagtail URLs in your patterns, set the `PUPUT_AS_PLUGIN` setting to `True`. This will avoid duplicate inclusion of Wagtail's URLs when you include Puput's URLs.
4. Include Puput's URLs in your patterns **before** Wagtail's URLs.

```
urlpatterns = [
    ...
    url(r'', include('puput.urls')),
    url(r'', include(wagtail_urls)),
]
```

5. Run `python manage.py migrate`.

Docker

If you want to run Puput in a Docker container please visit docker-puput for detailed instructions.

Editor's dashboard

Puput uses the default Wagtail CMS admin page in order to manage the content of the blog. It provides a powerful, clean and modern interface. Just open your browser at http://127.0.0.1:8000/blog_admin/.

This is how adding entry page looks:

Please visit Wagtail: an Editor's guide for further details of how to use Wagtail editor's dashboard.

Comments

Puput allows customize the comment system for your blog entries. Simply go to settings tab while editing blog properties and add the required parameters depending on which system you want to use.

Disqus

Set *Disqus api secret* and *Disqus shortname* with your project values and comments will be displayed in each blog entry. *Disqus api secret* is needed to retrieve the number of comments of each entry. If you don't need such data in your blog just fill *Disqus shortname* field.

Note: If you set *Disqus api secret* you need to install *tapioca-disqus* to access to the Disqus API

```
pip install tapioca-disqus
```

Feeds

Puput allows to customize the feeds for your blog entries. These options can be found in the settings tab while editing blog properties.

Feed description

Set *Use short description in feeds* to False if you want to use the full blog post content as description for the feed items. When set to True (by default), Puput will try to use the blog post's excerpt or truncate the body to 70 words when the excerpt is not available.

Extending Entry Page

Puput allows extend the `EntryPage` model. It provides two approaches to extend entries depending on the project requirements.

Multi-table inheritance

The easiest way to extend `EntryPage` model is using multi-table inheritance. Imagine if you need an special entry that needs a mandatory video url. You can write an entry model like this on `models.py` of your project:

```
from django.db import models
from puput.models import EntryPage


class VideoEntryPage(EntryPage):
    video_url = models.URLField()
    content_panels = EntryPage.content_panels + [
        FieldPanel('video_url')
    ]
```

You also need to modify `subpage_types` field of `BlogPage` model as by default is bounded to have only `EntryPage` as children. You can rewrite the above example with this:

```
from django.db import models
from puput.models import EntryPage, BlogPage
```

```
class VideoEntryPage(EntryPage):
    video_url = models.URLField()
    content_panels = EntryPage.content_panels + [
        FieldPanel('video_url')
    ]
BlogPage.subpage_types.append(VideoEntryPage)
```

This will create two independent tables on the database so you can create entries on your blog that are instances of `EntryPage` or `VideoEntryPage`.

Abstract base classes

Another approach to have an extension of entries is using abstract base classes inheritance method by inheriting from `EntryAbstract` instead of `EntryPage`. In the previous example, it's shown a blog with regular entries (`EntryPage`) and tv entries (`VideoEntryPage`). If you only want to have `VideoEntryPage` on your blog and create a simple table you need to extend `EntryAbstract` model on `models.py` of your project.

```
from django.db import models
from puput.abstracts import EntryAbstract
from wagtail.wagtailadmin.edit_handlers import FieldPanel


class VideoEntryAbstract(EntryAbstract):
    video_url = models.URLField()

    content_panels = [
        FieldPanel('video_url')
    ]

    class Meta:
        abstract = True
```

Warning: Do not import the `EntryPage` model in your `models.py` where defining the abstract extended model because it will cause a circular importation.

Registering entry extension

You have to register the model extension in `settings.py` adding `PUPUT_ENTRY_MODEL` with the path of the abstract model.

Following the previous example you have to add `PUPUT_ENTRY_MODEL` in your `settings.py` file:

```
PUPUT_ENTRY_MODEL = 'youproject.models.VideoEntryAbstract'
```

Migrations

If you extend `EntryPage` model you must migrate the database in order to the see the changes that you made on the model. However if you perform a `makemigrations` operation it will create a migration in `puput.migrations` of your local Puput module folder.

So you need to define a new path to store the changes made on `EntryPage` model extension. You have to use `MIGRATION_MODULES` for this purpose:

```
MIGRATION_MODULES = {'puput': 'youproject.puput_migrations'}
```

After run `makemigrations puput` migrations will appear on `puput_migrations` folder.

Note: It's recommended that the new initial migration represents the initial Puput migration in order to avoid conflicts when applying `migrate puput` command. A recommend way is run `makemigrations puput` **before** define Entry model extension on `settings.py` by setting `PUPUT_ENTRY_MODEL`.

Import your blog data

If you need to migrate a blog system to Puput we provide you a various tools to import your data.

Zinnia

1. Install zinnia-to-puput package and its dependencies `pip install zinnia-to-puput`
2. Add *zinnia2puput* to your `INSTALLED_APPS` in `settings.py` file.
3. Run the management command:

   ```
   python manage.py zinnia2puput
   ```

You can optionally pass the slug and the title of the blog to the importer:

```
python manage.py zinnia2puput --slug=blog --title="Puput blog"
```

Wordpress

1. Install wordpress-to-puput package and its dependencies `pip install wordpress-to-puput`
2. Add `wordpress2puput` to your `INSTALLED_APPS` in `settings.py` file.
3. Run the management command:

   ```
   python manage.py wp2puput path_to_wordpress_export.xml
   ```

You can optionally pass the slug and the title of the blog to the importer:

```
python manage.py wp2puput path_to_wordpress_export.xml --slug=blog --title="Puput blog
↪"
```

Blogger

1. Install blogger2puput package and its dependencies `pip install blogger2puput`
2. Add `blogger2puput` to your `INSTALLED_APPS` in `settings.py` file.
3. Run the management command:

   ```
   python manage.py blogger2puput --blogger_blog_id=Your BlogID --blogger_api_
   ↪key=Your APIKey
   ```

You can optionally pass the slug and the title of the blog to the importer:

```
python manage.py blogger2puput --slug=blog --title="Puput blog" --blogger_blog_
↪id=Your BlogID --blogger_api_key=Your APIKey
```

Sites structure

Multi blog site

The Wagtail default page architecture it allows to create a tree based CMS where editors could create multiple pages that are children from others. The Puput architecture also follows this philosophy but you can only create Blog pages as parents and Entry pages as children. Furthermore all Blog pages must have Root page as parent.

This has a powerful advantage so you can create separated sites with multiple blog instances. For instance, you could create a simple blog *http://www.example.com/blog/* and another one with videos (a videoblog) *http://www.example.com/tv/*.

Single blog site

A common case of use is having a site as a blog. In this case, Puput is also good for this purpose. If you have a site like *http://www.myblog.com* and you want that the root of the site will be blog page you can modify our Root page on site configuration (usually here) and select the desired blog page. So with this you will be able to go *http://www.myblog.com* instead of *http://www.myblog.com/blog/*.

Settings

Puput provides setting variables in order to customize your installation.

PUPUT_ENTRY_MODEL

Default value: `'puput.abstracts.EntryAbstract'` (Empty string)

String setting to define the base model path for Entry model. See *Extending Entry Page* for more details.

PUPUT_AS_PLUGIN

Default value: `False` (Empty string)

Boolean setting to define if you set Puput as a plugin of a previously configured Wagtail project.

PUPUT_USERNAME_FIELD

Default value: `'username'` (Empty string)

String setting to define the default author username field. Useful for people that are using a custom User model and/or other authentication method where an username is not mandatory.

PUPUT_USERNAME_REGEX

Default value: `'\w+'` (Empty string)

String setting to define the default author username regex used in routes. Useful for people that are using a custom User model.

Changelog

0.8 (2016-11-17)

- Add Travis CI integration and functional tests.
- Add Django 1.10 support.
- Minor template tweaks.

0.7 (2016-08-18)

- Add initial travis support.
- Add canonical url and social share tags in templates for SEO purposes.
- Allow to place Puput's blog at any sitemap level.
- Fix issue in entry comments update method.
- Add PageChooserPanel for related entries chooser.
- Improve flexibility on adding other comment systems.
- Minor bug fixes.

0.6 (2016-05-18)

- Fix issue when displaying entries without images.
- Fix css issues.
- Add django-compressor as project dependency.
- Improve tags visualization.

0.5.2 (2016-02-18)

- Removed django-endless-pagination which is no longer maintained and is not compatible with Django 1.9. Replaced by django-el-pagination.
- Category slug is now editable on snippets section.

0.5.1 (2016-02-16)

- Fix bug due a missing template tag.

0.5 (2016-02-12)

- Altered URL structure in order to have blog as Wagtail root page.
- Added Docker integration.
- Archive list is now collapsible.

0.4.1 (2016-01-19)

- Minor css bug fixes.

0.4 (2015-12-09)

- Added a fancy logo.
- Improved visualization of entries header images.
- Minor bug fixes.

0.3 (2015-11-15)

- Customizable username field in settings file.
- Improvements in the documentation.
- CSS cleanup. Added LESS file.
- Minor bug fixes.
- Added catalan translations.

0.2 (2015-09-22)

- Extensible entry model.

0.1 (2015-09-12)

- Initial release.

Marc Tudurí Cristina Hernandez Felipe Arruda Edu Herraiz David Valera Iker Cortabarría Carlos Salom Basil Shubin