## Contents

1 Features

2 Contents:

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Setup</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Editor’s dashboard</td>
<td>8</td>
</tr>
<tr>
<td>2.3 Comments</td>
<td>8</td>
</tr>
<tr>
<td>2.4 Feeds</td>
<td>9</td>
</tr>
<tr>
<td>2.5 Extending Entry Page</td>
<td>9</td>
</tr>
<tr>
<td>2.6 Import your blog data</td>
<td>11</td>
</tr>
<tr>
<td>2.7 Sites structure</td>
<td>12</td>
</tr>
<tr>
<td>2.8 Settings</td>
<td>12</td>
</tr>
<tr>
<td>2.9 Changelog</td>
<td>13</td>
</tr>
<tr>
<td>2.10 Authors</td>
<td>15</td>
</tr>
</tbody>
</table>
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
• Configurable default template color
• Social share of blog entries (facebook, twitter, linkedin and google plus)
Blog Example
My first Blog

Related Entries

Last Entries
My second blog entry
April 04, 2018

Popular Entries
My second blog entry
April 04, 2018

Categories
Blogging

Tags
Top
Pop

Archive
2018

Powered by Django, Wagtail and Pupa
2.1 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*.

2.1.1 Standalone blog app

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

```python
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:

```python
INSTALLED_APPS = (  
    ...  
    'wagtail.core',  
    'wagtail.admin',  
    'wagtail.documents',  
    'wagtail.snippets',  
    'wagtail.users',  
    'wagtail.images',  
    'wagtail.embeds',  
    'wagtail.search',  
    'wagtail.sites',  
    'wagtail.contrib.redirects',  
    'wagtail.contrib.forms',  
```

(continues on next page)
3. Add Wagtail’s required middleware classes to `MIDDLEWARE_CLASSES` in your Django settings.

```python
MIDDLEWARE_CLASSES = (  
    ...  
    'wagtail.core.middleware.SiteMiddleware',  
    'wagtail.contrib.redirects.middleware.RedirectMiddleware',  
)
```

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

```python
TEMPLATE_CONTEXT_PROCESSORS = (  
    ...  
    'django.template.context_processors.request',  
)
```

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

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

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

```python
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.

```python
urlpatterns = [  
    ...  
    path(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`:

```python
from django.conf import settings  
from django.conf.urls import url  
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  
    urlpatterns += static(settings.MEDIA_URL + 'images/', document_root=os.path.join(settings.MEDIA_ROOT, 'images'))
```

(continues on next page)
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.

### 2.1.2 Installation on top of Wagtail

0. This assumes that you have Wagtail >= 2.0 installed and you can access /admin; if this is not the case or you would like to use a newer version of Wagtail than is in the dependencies of puput, follow the steps below in a python venv:

```
pip install --upgrade pip
pip install wheel
pip install wagtail django-colorful django-el-pagination django-social-share
pip install --no-deps puput
wagtail start mysite
cd mysite
python manage.py migrate
python manage.py createsuperuser
```

1. If you haven’t already, install Puput and its dependencies via `pip install puput`.

2. In your Django settings (most commonly `settings/base.py` inside the wagtail directory), add the following to the `INSTALLED_APPS` following the wagtail section:

```
'wagtail.contrib.sitemaps',
'wagtail.contrib.routable_page',
'django_social_share',
'puput',
'colorful',
```

3. In the same file, also add the line `PUPUT_AS_PLUGIN = True` to the very bottom

4. In the same folder, add to `urls.py` near the top from `puput import urls as puput_urls` and just above `url(r'', include(wagtail_urls)), add `url(r'', include(puput_urls)),`

5. Run `python manage.py migrate` followed by `python manage.py runserver 0:8000` to start the server

6. To create a Puput blog navigate to the Wagtail admin interface at `127.0.0.1:8000/admin` and create a new child page of type `Blog`. Every blog post is then created as a child of this blog.

### 2.1.3 Docker

If you want to run Puput in a Docker container please visit `docker-puput` for detailed instructions.
2.2 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.

Note: If you want to edit the owner of an entry you need to install Wagtail >= 1.11.

2.3 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.

2.3.1 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.

```shell
pip install tapioca-disqus
```
2.4 Feeds

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

2.4.1 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.

2.5 Extending Entry Page

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

2.5.1 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:

```python
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:

```python
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`. 
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.

```python
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 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.
2.6 Import your blog data

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

2.6.1 Prerequisites

All importers need the lxml Python package, which has the prerequisites libxml2 and libxslt.
To install on Ubuntu:

```bash
sudo apt-get install libxml2-dev libxslt-dev
```

To install on CentOS or Red Hat:

```bash
sudo yum install libxml2-devel libxml++-devel libxslt-devel
```

2.6.2 Zinnia

1. Install zinnia-to-puput package and its dependencies
   ```bash
   pip install zinnia-to-puput
   ```
2. Add `zinnia2puput` to your `INSTALLED_APPS` in `settings.py` file.
3. Run the management command:
   ```bash
   python manage.py zinnia2puput
   ```
You can optionally pass the slug and the title of the blog to the importer:

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

2.6.3 Wordpress

1. Install wordpress-to-puput package and its dependencies
   ```bash
   pip install wordpress-to-puput
   ```
2. Add `wordpress2puput` to your `INSTALLED_APPS` in `settings.py` file.
3. Run the management command:
   ```bash
   python manage.py wp2puput path_to_wordpress_export.xml
   ```
You can optionally pass the slug and the title of the blog to the importer:

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

2.6.4 Blogger

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

### 2.7 Sites structure

#### 2.7.1 Multi blog site

The Wagtail default page architecture 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/](http://www.example.com/blog/) and another one with videos (a videoblog) [http://www.example.com/tv/](http://www.example.com/tv/).

#### 2.7.2 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](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](http://www.myblog.com) instead of [http://www.myblog.com/blog/](http://www.myblog.com/blog/).

### 2.8 Settings

Puput provides setting variables in order to customize your installation.

#### 2.8.1 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.

#### 2.8.2 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.

#### 2.8.3 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.
2.8.4 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.

2.9 Changelog

2.9.1 1.0.2 (2018-05-17)

• Add missing image.
• Add missing str methods.

2.9.2 1.0.1 (2018-04-19)

• Fix header image.

2.9.3 1.0 (2018-04-10)

• Add support for Django 2.0 and Wagtail 2.0. Drop Python 2.7 support.
• Add code and block quote options to the entries text editor.
• Improve default template visualisation.
• Configurable default template color.
• Add social sharing links.

2.9.4 0.9.2 (2018-02-13)

• Remove django-compressor dependency.
• Add German and Polish translations.

2.9.5 0.9.1 (2017-09-12)

• Add missing migration.

2.9.6 0.9 (2017-08-03)

• Fix issue that creates undesired migrations.
• Add Django 1.11 & Python 3.6 support. Drop Python 3.3 support.
• Improve RSS feeds generation.
• Fix issue with templatetags.
• Add Italian, Russian, Brazilian and French translations.
• Fix url resolution issues.
2.9.7 0.8 (2016-11-17)

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

2.9.8 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.

2.9.9 0.6 (2016-05-18)

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

2.9.10 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.

2.9.11 0.5.1 (2016-02-16)

• Fix bug due a missing template tag.

2.9.12 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.

2.9.13 0.4.1 (2016-01-19)

• Minor css bug fixes.
2.9.14 0.4 (2015-12-09)

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

2.9.15 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.

2.9.16 0.2 (2015-09-22)

- Extensible entry model.

2.9.17 0.1 (2015-09-12)

- Initial release.

2.10 Authors

- Marc Tudurí
- Cristina Hernandez
- Felipe Arruda
- Edu Herraiz
- David Valera
- Iker Cortabarría
- Carlos Salom
- Basil Shubin
- Milton Lenis
- Timothy Allen
- Pieter De Decker