django-nomadblog Documentation

Release 0.5

Hector Garcia

May 02, 2017

Contents

This is a basic Django application implementing the simplest form of a blogging system. It is capable of managing multible blogs and users. It has been written with an eye put on keeping modularity and flexibility as much as possible, so you won't find lots of goodies in the code (tagging, related posts, blogroll), but just a couple of features to help you start hacking it to your needs.

Contents:

CHAPTER 1

Features

- **Multiblog**: A simple project setting and a URL pattern is all we need to configure our Nomadblog installation for single or multiple blog management.
- **Multiuser**: The application allows one or many authors writing content to the same blog.
- **Simplicity and flexibility**: A post can have a category and an status assigned. That's it. Everything else is up to you and you will have to code it yourself, which means you control what is in your project, how it works, etc.
- **Context update functions**: Default Nomadblog views are only a thin wrapper that call their respective `_ctxt` related functions (which have the business logic) and return the response. The use of this mechanism is better explained in the section Custom view wrappers.
- **Complete example projects**: As usual with my open sourced Django projects, I include two example project folders, one for a single blog installation (`single_blog_example`) and another one for multiblog (`multiple_blogs_example`), where you can see all the things explained here in practice.

CHAPTER 2

Installation

The package is listed in the Python Package Index. You can use your favorite package manager like `easy_install` or `pip`:

```
pip install django-nomadblog
```

Or, you can clone the latest development code from its repository:

```
git clone git@github.com:nabucosound/django-nomadblog.git
```

CHAPTER 3

Configuration

Add nomadblog to the INSTALLED_APPS setting of your settings.py:

```
INSTALLED_APPS = (
    ...
    'nomadblog',
)
```

If you use South you can run the included migrations:

```
./manage.py migrate nomadblog
```

If you don't, use your own migration tool or simply syncdb:

```
./manage.py syncdb
```

Include this two lines of code in your root urls.py:

```
# Put it somewhere in the beginning of the file
from django.conf import settings
MULTIBLOG = getattr(settings, 'NOMADBLOG_MULTIPLE_BLOGS', False)

# Add this pattern into your url conf
urlpatterns = patterns('',
    ...
    url(r'^blog/', include('nomadblog.urls')) if not MULTIBLOG \
        else (r'^blogs/(?P<blog_slug>[-\w]+)/', include('nomadblog.urls')),
)
```

You can change the blog/ or blogs/ initial part but do not modify (?P<blog_slug>\w+), because it is used by the app to differenciate which blog is being accessed, in case multiblog is used.

Settings

Multiblog

Define the variable `NOMADBLOG_MULTIPLE_BLOGS` in your project settings as `True` if you want a multiple blog configuration:

```
NOMADBLOG_MULTIPLE_BLOGS = True
```

Default Post model

By default, `django-nomadblog` uses the `Post` model, but you can extend it with your own one, that will be then used by the app views:

```
POST_MODEL = 'yourapp.models.YourExtendedPostModel'
```

CHAPTER 4

Overriding templates

`django-nomadblog` uses different templates to list posts, show a post details, list categories or list posts by category. You will want to override these templates, to add your layout, design and own stuff. Create a new `nomadblog` template folder where your project can find it and copy the templates found on the `templates/nomadblog` directory (like `list_posts.html` or `show_post.html`) or, if you want to be quicker, just copy the entire `templates/nomadblog` folder.

CHAPTER 5

Passing parameters to views

Views receive a number of parameters that are used to specify, change or override different parts of the app. If you take a look at the Nomadblog default views you will see how flexible Nomadblog is intended to be.

In order to pass parameters to Nomadblog views, you must first of all create a copy of the `urls.py` file in the Nomadblog app:

```
cp /path/to/django-nomadblog/urls.py /path/to/project/yourapp/blog_urls.py
```

Then point to it changing your project root URL pattern:

```
urlpatterns = patterns('',
        ...
    (r'^blog/', include('yourapp.blog_urls')) if not \
    NOMADBLOG_MULTIPLE_BLOGS else (r'^blogs/(?P<blog_slug>\w+)/', \
    include('yourapp.blog_urls')),
)
```

You can now modify your urlconf, like passing parameters to view functions with the `kwargs`. See the `website/blog_urls.py` file in the example projects to check out a few examples.

CHAPTER 6

Custom view wrappers

If you want to extend functionality beyond the basic logic behind a Nomadblog view, you can call, from your wrapper view function, one of the `_ctxt` functions defined in `views.py` directly with your context. Passing a `RequestContext` to the function will update it with the expected values needed for rendering the response. If you do not pass any `RequestContext` object, a new one is created and returned.

Basically the idea behind having the business logic separated from template context population is that you can have the basic functionality of the action performed in the blog (get a list of posts, show the contents of a post) isolated and DRY, and add or modify business logic to your wrapper view.

I wrote a post trying to explain better this approach. Also, the four Nomadblog actions represented by their four view functions — `list posts`, `show post`, `list categories`, `show post by category` — in the `views.py` code are actually the best examples to implement your own wrapper.

CHAPTER 7

Reverse urls

Reverse URLs in templates will vary depending on your multiblog configuration. Nomadblog views add a `multiblog` flag in the context to use the right `url` template tag parameters. Take, for instance, this sampe code from `show_post.html`:

```
{% if multiblog %}
<a href="{% url list_posts_by_category
bloguser.blog.slug post.category.name %}" class="link-categories">
{{ post.category.name }}</a>
{% else %}
<a href="{% url list_posts_by_category post.category.name %}"
class="link-categories">{{ post.category.name }}</a>
{% endif %}
```

You probably won't need this if you are using your own templates, because you will set up your templates in advance.