django-sluggable Documentation

Release dev

May 23, 2013

CONTENTS

django-sluggable is a library to manage your slugs and redirect old slugs to a new one. With this library, you will have the plain history of your operations.

You can report bugs and discuss features on the issues page.

CHAPTER ONE

REFERENCE

For further details see the reference documentation:

1.1 Installation

Either check out django-sluggable from GitHub or to pull a release off PyPI

```
pip install django-sluggable
```

1.2 Usage

1.2.1 Integrated in an application

To use django-sluggable we will provide a basic application in this section.

Consider having the following `models.py`:

```
# users/models.py

class User(models.Model):
    username = models.CharField(max_length=150)
```

Now you want urls like `/users/<username>` but also keeping your SEO when a specific user is changing his username: we want a permanent redirection between the old username and the new one.

In `models.py`, we will define a decider model which will store all usernames:

```
# users/models.py
from sluggable.models import Slug


class UserSlug(Slug):
    class Meta:
        abstract = False
```

In the case of our `User` class the slug is basically the username of the user, so we will change the type of the `username` field.

```
# users/models.py
from sluggable.fields import SluggableField
```

```
class User(models.Model):
    username = SluggableField(decider=UserSlug)

    def __unicode__(self):
        return self.username
```

Now you have your sluggable model, let's play with the API, by adding our first member in the console:

```
In [1]: from users.models import User, UserSlug
In [2]: user = User.objects.create(username="thoas")
```

When you are creating a new `User` it will also create a linked model by using the contenttypes framework of Django:

```
In [3]: user_slug = UserSlug.objects.get(slug="thoas")
In [4]: user_slug.redirect
False
```

With this `UserSlug` you can now track every username changes by your users.

Remember your first created user right? We will change its username:

```
In [5]: user.username = 'oleiade'
In [6]: user.save()
```

You new username is now your primary username and you will be able to provide a permanent redirection between the old one and new one:

```
In [7]: user_slug = UserSlug.objects.get(slug="oleiade")
In [8]: user_slug.redirect
False
In [9]: old_slug = UserSlug.objects.get(slug="thoas")
In [10]: old_slug.redirect
True
```

If you are accessing an old slug, you can also retrieve the current one at any time:

```
In [11]: old_slug.current
<Slug thoas for oleiade>
```

If you do not have a `Slug` instance, no problem you can use the default manager for that:

```
In [12]: Slug.objects.get_current(user)
<Slug oleiade for oleiade>
```

1.2.2 Work with class-based views

Now you know how to manipulate your users, we will add real world examples in an real application.

Let's begin with the `views.py` file.

In this section, we will only use Class-based views so if you are not familiar with them, go check them they are awesome:

```
# users/views.py
from django.views import generic

from users.models import User
```

```
class UserDetailView(generic.Detail):
    model = UserSlug
    context_object_name = 'slug'
    slug_field = 'username'
    template_name = 'users/detail.html'


# users/urls.py
from users import views


urlpatterns = patterns('',
    url(r'^users/(?P<username>\w+)/$',
        views.UserDetailView.as_view(),
        name='user_detail'),
)
```

So we have defined a pretty standard view to show an user with its username, so boring duh?

The interesting part is the redirection provided by django-sluggable, let's rewrite `UserDetailView.get`:

```
# users/views.py
from django.views import generic
from django.shorcuts import redirect

from users.models import User


class UserDetailView(generic.Detail):
    model = UserSlug
    context_object_name = 'user'
    slug_field = 'username'
    template_name = 'users/detail.html'

    def get(self, request, *args, **kwargs):
        obj = self.get_object()

        # The slug retrieved is a redirection to a new one
        if obj.redirect:

            # Retrieve the current slug used
            current = obj.current

            return redirect('user_detail', username=current.slug)

        # Retrieve the real object affected to the slug
        self.object = obj.content_object

        context = self.get_context_data(object=self.object)

        return self.render_to_response(context)
```

Wait? `UserDetailView.get` is big.

Let's rewrite it with django-multiurl to dispatch our slug management between multiple views.

With this new method, we don't have to rewrite `UserDetailView.get` anymore:

```
# users/views.py
```

```
from django.views import generic

from users.models import User, UserSlug

class UserDetailView(generic.Detail):
    model = User
    context_object_name = 'slug'
    slug_field = 'username'
    template_name = 'users/detail.html'


class UserRedirectView(generic.RedirectView):
    permanent = True

    def get_redirect_url(self, username):
        slug = get_object_or_404(UserSlug.objects.filter(redirect=True), slug=username)

        return reverse('user_detail', args=(slug.current.slug,))
```

But we have to rewrite our `urls.py` file to use django-multiurl:

```
# users/urls.py

from multiurl import multiurl, ContinueResolving

from django.http import Http404

from users import views

urlpatterns = patterns('',
    multiurl(
        url(r'^users/(?P<username>\w+)/$',
            views.UserDetailView.as_view(),
            name='user_detail'),
        url(r'^users/(?P<username>\w+)/$',
            views.UserRedirectView.as_view(),
            name='user_redirect'),
        catch = (Http404, ContinueResolving)
    )
)
```

1.2.3 Hidden features

How know if the slug has changed?:

```
In [1]: user = User.objects.create(username="thoas")
In [2]: user.slug_changed
False
In [3]: user.slug = 'oleiade'
In [4]: user.slug_changed
True
```

How to know if a slug is available or not?:

```
In [1]: user = User.objects.create(username="thoas")
In [2]: UserSlug.objects.is_slug_available('thoas')
False
In [3]: user.slug = 'oleiade'
In [4]: user.save()
In [5]: UserSlug.objects.is_slug_available('thoas')
False
```

If you are providing an optional `obj` parameter which has the slug:

```
In [6]: UserSlug.objects.is_slug_available('thoas', obj=user)
True
```

Restore previous slug and remove redirections:

```
In [7]: UserSlug.objects.update_slug(user, 'thoas', erase_redirects=True)
```

1.3 Architecture

...

1.4 Indices and tables

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

CHAPTER

TWO

ISSUES

For any bug reports and feature requests, please use the Github issue tracker.