django-nomad-notifier Documentation

Release 1.4.6

Hector Garcia

March 11, 2016

CHAPTER 1

Overview

This Django app provides a way to implement a **notification system** for the users of web apps that must receive updates from the site activity through the following channels:

- **Email notifications**: usually *transactional emails* (messages like “Hi Frankie, Jimbo is now following you” or “You have unlocked a new level”) but any type of email in practice.
- **Web notifications**: displayed/listed on UI (“ala” social apps such as Facebook or Google+). A list of notifications that can be marked as read.

Contents:

1.1 Installation

1.1.1 Dependencies

- django-model-utils: https://github.com/carljm/django-model-utils/

1.1.2 Initial setup

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

```
pip install django-nomad-notifier
```

Or, you can clone the latest development code from its repository and add the folder to your python path. For example, if you use virtualenvwrapper:

```
git clone git@github.com:Nomadblue/django-nomad-notifier.git
add2virtualenv django-nomad-notifier/
```

Add `notifier` to the `INSTALLED_APPS` setting of your `settings.py`:

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

We use migrations to nicely update our database with the `Notification` model schema. But **wait**! Since there are models that depend on custom user (`settings.AUTH_USER_MODEL`), we cannot provide the migrations with the package. Instead, our policy at Nomadblue is to create them out of the scope in another place.

Prior to Django 1.7, there was South . So, for example, in an app called `website` it would be:

```
SOUTH_MIGRATION_MODULES = {
    'notifier': 'website.notifier_migrations',
}
```

From your project root, create the initial migration and apply it:

```
python manage.py schemamigration notifier --initial
python manage.py migrate notifier
```

In Django 1.7 or beyond, where South was incorporated as part of the core, the equivalent setting is:

```
MIGRATION_MODULES = {
    'notifier': 'website.notifier_migrations',
}
```

From your project root, create the initial migration and apply it:

```
python manage.py makemigrations notifier
python manage.py migrate notifier
```

If you prefer not to use migrations, can sync with the Django command:

```
python manage.py syncdb
```

Finally, if you want to enable email notifications (disabled by default), you **must** set the following in your `settings.py`:

```
SEND_EMAIL_NOTIFICATIONS = True
```

1.2 Usage

`django-nomad-notifier` is not functional by itself. You must do your homework to get it running. This app is meant to provide only the essential parts for the sole purpose of exposing a model and a mixin. Those minimal code helps you build **notification models** that will inherit the functionality and will hold the information needed in order to send email notifications (and show them on web as well).

What we do at Nomadblue, for instance, is adding a new app `notifications` that will store the models, templates, and any other related files.

1.2.1 Creating notification models

You must do three things to successfully create a new type of notification:

- Subclass `models.Notification` to get the model fields and basic methods.
- Use the `NotificationMixin` to get the basic methods.
- Override some methods to provide the minimal functionality.

For this documentation we are going to create a simple **welcome message** notification that will be attached to a Django `User` model whenever a new user is created (i.e. a user signs up). It will be a notification that will be sent via email and also displayed on a web notifications list.

1.2.2 Subclassing `models.Notification`

First of all, the code for the impatient:

```
class WelcomeNotification(Notification, OurNotificationMixin, NotificationMixin):
    """Welcomes the user after signup"""

    # This is the model instance that the notification will reference to. In this
    # particular case, it has to be the user model who just signed up.
    obj = models.OneToOneField(settings.AUTH_USER_MODEL)

    # Template for web notification list
    web_noti_tmpl = 'notifications/includes/welcome_notification_list_item.html'

    # Templates for email notification
    email_subject_tmpl = 'notifications/email/welcome_noti_subject.txt'
    email_plaintext_body_tmpl = 'notifications/email/welcome_noti_plaintext_body.txt'
    email_html_body_tmpl = 'notifications/email/welcome_noti_html_body.html'

    def get_obj_url(self):
        """
        This method is used to specify URL to redirect user after
        notification is been cleared (a.k.a 'mark as read')
        """
        return self.obj.get_absolute_url()
```

As you can see, we must provide the template paths for the two notification types (web and email), the `get_obj_url` method (used in the `views.ClearNotificationView`) and, the most important, adding the `NotificationMixin` so our model inherits the methods defined there and which will help us sending the email notification.

But wait! You surely have taken into account on another mixin we are using, `OurNotificationMixin`. This mixin, whose code we provide below, holds some other **mandatory** methods we must implement if we want to send notifications via email:

```
class OurNotificationMixin(object):
    from_email = settings.DEFAULT_FROM_EMAIL

    def get_email_headers(self):
        # Replace 'Your name' with your real name or project name
        return {
            'From': 'Your name <%s>' % self.from_email,
        }

    def get_recipients_list(self):
        # Here we include only the notification user email but
        # in fact the recipient/s could be whichever.
        return [self.user.email]
```

These two methods are conveniently stored in a mixin because tipically we end up creating many notification models with different templates but the same common functionality.

Finally, we need to create our web and email templates. The property names are rather self-descriptive, but here it is defined what must each template contain:

- `web_noti_tmpl`: template that renders the web notification snippet (each item on a notification list).
- `email_subject_tmpl`: template that contains the email subject (e.g. “Thanks for signing up”)
- `email_plaintext_body_tmpl`: template containing the plaintext version of your email.

- `email_html_body_tmpl`: template containing the HTML version of your email.

1.2.3 Sending email notifications

OK, now what? As the example we are describing is a notification message a brand new user receives after the registration process in our site, we want firstly to send the email. So, we could for instance create a new instance of our `WelcomeNotification` from our signup view:

```
try:
    user.welcomenotification  # In case user is re-visiting the view by mistake
except WelcomeNotification.DoesNotExist:
    WelcomeNotification.objects.create(user=user, obj=user)
```

Last step is to make the call to the `NotificationMixin.send_notification_email` method. We fancy using the `post_save` signal here:

```
def send_email_notification(sender, instance, created, **kwargs):
    if created:
        instance.send_notification_email()

post_save.connect(send_email_notification, sender=WelcomeNotification)
```

1.2.4 Displaying web notifications

What about showing a list of notifications in the user interface, where they can review them and clear them (mark as read)? We can go straight using the urls and views powered in the app. Include the calls in your root `urls.py`:

```
url(r'^notifications/', include('notifier.urls')),
```

Assuming that you have already created your template (its path stored in `web_noti_tmpl`), if you visit `http://localhost:8000/notifications/` you should see a list of notifications. Of course you can go ahead and override the `notifier/notifications_list.html` template.

1.2.5 Types of notifications

In our previous example we created a web and email notification, i.e. a notification that is sent via email and displayed on you website. But perhaps you want to send an email but not show the notification via web. No problem! The `Notification` object supports specification of types of notifications with its `noti_type` field. Adjusting the creation of the notification above, we would end up with:

```
try:
    user.welcomenotification  # In case user is re-visiting the view by mistake
except WelcomeNotification.DoesNotExist:
    WelcomeNotification.objects.create(user=user, obj=user, noti_type=WelcomeNotification.EMAIL_NOTI
```

1.3 Settings

1.3.1 SEND_EMAIL_NOTIFICATIONS

Default: `False`

Set this to `True` to enable the app to send email notifications. It is disabled by default to avoid unwanted "surprises" (e.g. sending emails to real users by mistake), so we force you to explicitly set it up.

1.3.2 NOTI_FAIL_SILENTLY

Optional setting Default: `False`

If there is an error sending a notification via email, the SMTP exception is captured and stored in the object. By default, the exception is thrown and the application fails, but if setting is present and set to `True`, it will fail silently and not crash. It is sometimes for production servers, where we do not want errors to break our user experiences.