django-notifier Documentation

Release 0.7

Siddharth Doshi

August 19, 2014

Contents

django-notifier is a django app to send notifications and manage preferences and permissions per user and group.

It can support multiple methods of sending notifications and provides methods to manage user preferences as well as group settings for notifications. *email* notifications are supported out of the box using django's email settings, additional methods of sending notification (Backends) can be added as required to support *SMS*, *phone* or even *snail mail*.

The general idea is to reduce the overhead of setting up notifications for different actions (payment processed, new membership, daily update) and making it easy to use the same backend for all kinds of notifications.

CHAPTER 1

Dependecies

django-notifier supports Django 1.4 and later on Python 2.7 and later. Earlier versions of Python and Python 3 have not been tested.

CHAPTER 2

Contents

2.1 Installation & Setup

2.1.1 Installation

Install via `pip`.

To download latest dev version:

```
$ pip install git+git://github.com/scdoshi/django-notifier.git
```

From PyPI (latest release):

```
$ pip install django-notifier
```

2.1.2 Setup

1. Add `notifier` to INSTALLED_APPS in your django settings file.

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

2. Settings

   If you are going to use any custom backends to send notifications, add the setting NOTIFIER_BACKENDS to your settings file. If this setting is not defined, only the EmailBackend is used by default.

   ```
   NOTIFIER_BACKENDS = (
       'notifier.backends.EmailBackend',
       'path.to.custom.backend.CustomBackend',
   )
   ```

3. Run `syncdb` or `migrate` (if using South) to create the necessary tables in the database.

   ```
   $ python manage.py syndb
   ```

   If using South:

```
$ python manage.py migrate
```

2.1.3 Terminology

- `Notification`: A message that can be sent from the system to users (payment declined, email verification).
- `Backend`: A way to send notifications to users (email, SMS etc.)

2.2 Quickstart

A quick primer on how to start sending notifications.

2.2.1 Create Notification

1. Use the `create_notification` shortcut method to create a new type of notification.

   Convention is to declare all notifications for an app in `notifications.py` in the app directory.

   ```
   from notifier.shortcuts import create_notification
   create_notification('card-declined')
   ```

   This will create a notification called 'card-declined'.

2. Create templates for notifications you want to be sent.

   For the `email` backend, the required templates for the 'card-declined' notification will be:

   - `notifier/card-declined_email_message.txt`
   - `notifier/card-declined_email_subject.txt`

   These can be placed in the app's templates directory.

   The `Site` and `User` models are available as context for the templates. Different backends may use a different number and format of templates, but the convention is:

   ```
   <notification-name>_<backend>_<optional_descriptor>.txt
   card-declined_email_subject.txt
   ```

2.2.2 Send Notification

```
from notifier.shortcuts import send_notification
send_notification('card-declined', [user1, user2])
```

2.3 Backends

Backends are classes that define how a notification will be sent via a particular method. The `email` backend works out of the box using the django email settings. Other custom backends can be written to support other methods of sending notifications.

2.3.1 Email Backend

The email backend sends email notifications using Django's default email settings. Two templates are required per notification to form the email:

- Subject: `notifier/<notification-name>_email_subject.txt`
- Message: `notifier/<notification-name>_email_message.txt`

The templates already have the User and the Site objects passed in as context variables, additional context variables can be passed in using the send method for the notification. For e.g.

```
extra_context = {
    'var1': value1,
    'var2': value2
}

send_notification('notification-name', [user1, user2], extra_context)
```

Customization

The EmailBackend can be extended if the standard backend does not meet the requirements. For example, to add a users current membership status or some other such custom object to the context for all notifications:

```
from notifier.backends import EmailBackend
class CustomEmailBackend(EmailBackend):
    def send(self, user, context=None):
        if not context:
            context = {}
        context.update({
            'membership': user.membership_set.latest()
        })
        return super(CustomEmailBackend, self).send(user, context)
```

2.3.2 Custom Backend

A completely custom backend can be written to support any method of sending a notification.

Custom backends should be extended from the `BaseBackend` class. The `send` method in a backend class deals with how the actual message is sent.

The template used by default will be `notifier/<notification-name>_<backend-name>.txt`.

BaseBackend

class `notifier.backends.`**`BaseBackend`**(*notification*, **args*, ***kwargs*)

Example

An example of a custom backend to send SMS messages via Twilio.

```
from django.conf import settings
from django.template.loader import render_to_string
from notifier.backends import BaseBackend
from twilio.rest import TwilioRestClient
```

```
TWILIO_ACCOUNT_SID = getattr(settings, 'TWILIO_ACCOUNT_SID', None)
TWILIO_AUTH_TOKEN = getattr(settings, 'TWILIO_AUTH_TOKEN', None)
TWILIO_FROM_NUMBER = getattr(settings, 'TWILIO_FROM_NUMBER', None)
client = TwilioRestClient(TWILIO_ACCOUNT_SID, TWILIO_AUTH_TOKEN)

# self.template = 'notifier/<notification-name>_sms-twilio.txt'

class TwilioBackend(BaseBackend):
    name = 'sms-twilio'
    display_name = 'SMS'
    description = 'Send via SMS using Twilio'

    def send(self, user, context=None):
        super(TwilioBackend, self).send(user, context)

        message = render_to_string(self.template, self.context)

        sms = client.sms.messages.create(
            to=user.phone,
            from_=TWILIO_FROM_NUMBER,
            body=message
        )
        return True
```

2.4 Notifications

Declare notifications in `notifications.py`. These will be detected during syncdb or migrate and available to send via the different backends.

2.4.1 Functions

Create Notification

`notifier.shortcuts.`**`create_notification`**(*name*, *display_name=None*, *permissions=None*, *backends=None*, *public=True*)

Arguments

name notification name, unique (string)

display_name notification display name, can be non-unique (string)

permissions list of permission names or objects

backends list of backend names or objects

public (boolean)

Returns Notification object

Send Notification

`notifier.shortcuts.`**`send_notification`**(*name*, *users*, *context=None*)

Arguments

name notification name (string)

users user object or list of user objects

context additional context for notification templates (dict)

Returns

None

2.5 Preferences

Preferences about notifications can be set per user and per group.

2.5.1 Group Preferences

Per group preferences can be set so that if a user belongs to any group that has the the preference set to True, then a notification is sent, unless it is overridden by user preference.

Group preferences are stored in the `GroupPrefs` model.

`notifier.models.GroupPrefs`

2.5.2 User Preferences

Per user preferences override per group preferences. The user preference for a notification can only be set if the user has all the permissions required for that notification.

User preferences are stored in the `UserPrefs` model.

`notifier.models.UserPrefs`

2.5.3 Set Preferences

```
# User
notification_obj.update_user_prefs(user_obj, {'email': True, 'backend2': False})
# Group
notification_obj.update_group_preference(group_obj, {'email': True, 'backend2': False})

# or use a shortcut method
from notifier.shortcuts import update_preferences
# User
update_preferences('notification-name', user_obj, {'email': True, 'backend2': False})
# Group
update_preferences('notification-name', group_obj, {'email': True, 'backend2': False})
```

`notifier.shortcuts.`**`update_preferences`**(*name*, *user*, *prefs_dict*)

Arguments

name notification name (string)

user user or group object

prefs_dict dict with backend obj or name as key with a boolean value.

e.g. {'email': True, 'sms': False}, {email_backend_obj: True, sms_backend_obj: False}

Returns

dict with backend names that were created or updated. values that do not require change are skipped

e.g. {'email': 'created', 'sms': updated}

2.5.4 Clear Preferences

There is a shortcut method to clear all *user* preferences (set preferences back to default)

```
from notifier.shortcuts import clear_preferences
clear_preferences([user1, user2])
```

notifier.shortcuts.**clear_preferences**(*users*)

Arguments

users user object or list of user object

Returns

None

2.5.5 Form

django-notifier has a formset that includes a form for every notification along with checkboxes for every backend for that notification. This can be used in a view to allow the users to set notification preferences.

```
notifier.forms.NotifierFormSet
```

An example of customizing the formset in django templates:

```
{% load attribute %}

{% if formset.forms %}
<form id="notification_form" name="input" method="post" autocomplete="off" class="form">
    {% csrf_token %}
    {{ formset.management_form }}
    <div class="form_item table"><table>
        <tr>
            <th>Notification</th>
            {% for method in formset.dm %}
                <th>{{ method.display_name }}</th>
            {% endfor %}
        </tr>
        {% for form in formset %}
            <tr>
                <td>
                    <div class="form_label">{{ form.title }}</div>
                </td>
                {% for method in formset.dm %}
                    <td>
                        {% with field=form|attr:method.name %}
                            {% if field %} {{ field }} {% endif %}
                        {% endwith %}
                    </td>
                {% endfor %}
            </tr>
        {% endfor %}
    </table></div>
```

```
    <div class="form_item">
        <div class="two-button">
            <input type="submit" value="Save">
        </div>
    </div>
</form>
{% else %}
<p>There are no notifications that can be configured.</p>
{% endif %}
```

2.6 Change Log

2.6.1 0.7

- BREAKING - NotificationManager.update_user_prefs() removed.
- Notification.update_group_prefs() on added to allow setting group preferences.
- shortcuts.update_prefrences() is a shotcut to set both user and group preferences.
- 'read' field for SentNotifications.

    If your notification method supports read reciepts, you can use this field to keep track of it. Not used anywhere right now, but we could add webhook support to mark messages as read.

CHAPTER 3

Contribute / Report Errors

To contribute to the code or docs, please visit the github repository at

https://github.com/scdoshi/django-notifier

To report errors, please create a new issue at

https://github.com/scdoshi/django-notifier/issues

CHAPTER 4

To Do

- Add notification support for non-user addresses (emails, phones etc.)
- Functions to output all email addresses for particular notification
- Ways to send a notification to everyone in a notification list

CHAPTER 5

Indices and tables

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