django-heythere Documentation

Release 1.0.1

Kenneth Love

December 23, 2015

Contents

`django-heythere` is a small package for handling notifications to users. It doesn't handle every situation imaginable but it works great with custom users (even those with email fields not named `email`) and supports any email backend that Django does. It doesn't require you to send emails to users, either, if that's not your thing.

Currently tested against Python 2.6, 2.7, and 3.3 with Django 1.5, 1.6, and 1.7.

CHAPTER 1

Installation

1. `pip install django-heythere` or download from github and run `python setup.py install`.
2. Add `'heythere'` to your `INSTALLED_PACKAGES` in `settings.py`.
3. **OPTIONAL** Set up your special *configuration*.
4. `python manage.py syncdb`
5. The rest is up to you!

CHAPTER 2

Configuration

`django-heythere` is controlled almost entirely through a dictionary in `settings.py`. Here's the default settings:

```
NOTIFICATIONS = {
    'DEFAULT': {
        'persistent': True,  # stays until dismissed
        'send_as_email': False,  # send as email
        'headline_template': '{{headline}}',  # Django template for headline
        'body_template': '{{body}}',  # Django template for body
        'email_field': 'email'  # Assume field named 'email' is user's email
    }
}
```

Just provide your own dictionary if you want to override anything. Each type of notification is another key in the `NOTIFICATIONS` dictionary.

- `persistent`: Whether or not notifications are marked as inactive once emailed to a user.
- `send_as_email`: Whether or not to send this kind of notification as an email.
- `headline_template`: A Django template string that'll be rendered with a context dictionary for the headline.
- `body_template`: A Django template string that'll be rendered with a context dictionary for the body.
- `email_field`: The field on your user model that holds the user's email address.

Along with those configuration options, the `Notificiation` object has a few special methods and properties.
create_notification

`Notification.objects.create_notification(user, notification_type, headline, body)`

- `user` is the user object that should recieve the notification.
- `notification_type` is the key in your `NOTIFICATIONS` dict for the type of notification you want to send.
- `headline` and `body` are dictionaries of variables that you want to parse for the `headline_template` and `body_template`. They will be stored in `headline_dict` and `body_dict`, respectively, on the model instance.

clear_all

`Notification.objects.clear_all(<user>)` marks all unread notifications for a user as having been read.
send_all_unsent

`Notification.objects.send_all_unsent()` finds all unread notifications that are allowed to be sent as emails and...sends them. If they're marked as being non-persistent, they'll be marked as no longer active, too. **read**

`notification.read()` marks a `Notification` instance as having been read. **send_email**

`notification.send_email()` sends a `Notification` instance to its user.

CHAPTER 3

Admin and Management

There is a basic admin provided with the app that provides a new **action** and **button** on the changelist page. The button sends all unsent notifications as email (assuming the notification type allows that) and the action does the same for whichever notifications have been selected.

There is also a **management command** to send all unsent notifications (again, ignoring those whose notification type doesn't allow them to be sent). You can run this command with `./manage.py send_unsent_notifications`.

CHAPTER 4

Changelog

CHAPTER 5

Indices and tables

- genindex
- modindex
- search