django-campaign Documentation

Release 0.3.0-pre

Arne Brodowski

May 18, 2016

Contents

Newsletter and campaign management for the Django webframework.

Django-campaign is an application for the Django webframework to make sending out newsletters to one or more groups of subscribers easy. If you need to send newsletters to thousands of subscribers it is easy to integrate django-campaign with django-mailer or some email sending providers through their APIs.

Some of the core features are:

- Multipart Emails made easy - just add a plain-text *and* a html-template.
- Full control over the Subscriber-Model and therefore the template context used to render the mails.
- Add context processors to add whatever you need to a mail template based on the recipient. This makes it easy to personalize messages.
- Allow viewing of the newsletters online and add a link to the web version to the outgoing emails.
- simple and optional subscribe/unsubscribe handling

CHAPTER 1

Contents

1.1 Installation

Install django-campaign with `easy_install` or `pip` or directly from the GitHub repository.

Then add `campaign` and 'django.contrib.sites' to your `INSTALLED_APPS` setting:

```
INSTALLED_APPS = (
    ...
    'django.contrib.sites',
    'campaign',
    ...
)
```

Add an entry to your URL-conf. Using `campaign` here is a matter of taste, feel free to mount the app under a different URL:

```
urlpatterns += patterns('',
    (r'^campaign/', include('campaign.urls'))
)
```

4. Then run `manage.py syncdb` to create the neccessary database tables.

1.2 How to send Newsletters with django-campaign

Here is a very brief overview how to use django-campaign:

- If you plan to send out different Newsletters to different SusbcriberLists, it might be a good idea to create a Newsletter objects through the Django Admin.
- Setup one or more SubscriberList objects in the Django Admin interface or programmatically.
- Create at least one MailTemplate object through the Django Admin or programmatically.
- Create a Campaign object and assign the corresponding MailTemplate object, one or more SubscriberLists.
- Send the Campaign through the Django Admin or programmatically.

1.3 Available Settings

Here is a list of all available settings of django-campaign and their default values. All settings are prefixed with `CAMPAIGN_`, although this is a bit verbose it helps to make it easy to identify these settings.

1.3.1 CAMPAIGN_BACKEND

Default: `'campaign.backends.send_mail'`

The backend used for the actual sending of the emails. The default backend `campaign.backends.send_mail` uses Django's built-in e-mail sending capabilities.

Additionally the following backend is available:

- `'campaign.backends.mandrill_api'`: Uses Mandrill for sending the e-mails.

Please see the *backend docs* about implementing your own backend.

1.3.2 CAMPAIGN_CONTEXT_PROCESSORS

Default: `('campaign.context_processors.recipient',)`

Similar to Django's Template Context Processors these are callables which take a Subscriber object as their argument and return a dictionary with items to add to the Template Context which is used to render the Mail.

The following processors are availble within the django-campaign distribution:

- `recipient`: Implements the old default behaviour and adds the Subscriber object (a Django Model instance) to the mail context under the name *recipient*.
- `recipient_dict`: Serializes the Subscriber object to a dict before adding it to the context. This is neccesary, if you want to pass per recipient variables to a remote service. Use this a basis for your own campaign context processors.

1.3.3 CAMPAIGN_SUBSCRIBE_CALLBACK

Default: `None`

If CAMPAIGN_SUBSCRIBE_CALLBACK is configured the handling of newsletter subscriptions via django-campaign will be enabled.

You have to supply either a callable or an import-path to a callable, which accepts an email address as argument and returns either True or False to indicate if the action was performed successfully.

Example settings.py:

```
CAMPAIGN_SUBSCRIBE_CALLBACK = "myproject.newsletter.utils.subscribe"
```

Example implementation of the callback in your app:

```
def subscribe(email):
    s,c = Subscriber.objects.get_or_create(email=email,
                                           defaults={'newsletter': True})
    return True
```

It's up to you to decide where to store the Subscribers, as django-campaign is completely agnostic in this point. One or more Subscriber models can be defined via the admin interface for the *SubscriberList* model.

1.3.4 CAMPAIGN_UNSUBSCRIBE_CALLBACK

Default: `None`

Please see *CAMPAIGN_SUBSCRIBE_CALLBACK* above and replace subscribe with unsubscribe.

1.4 Backends

To decouple the actual sending of the e-mails from the application logic and therefore make django-campaign more scaleable version 0.2 introduces a concept of backends which encapsulate the whole process of sending the e-mails.

1.4.1 Writing your own Backend

Backends for django-campaign must adhere to the following API. Some of the methods are mandatory and some are optional, especially if you inherit from the `base` backend.

The basic structure of a backend is as follows:

```
from campaign.backends.base import BaseBackend

class ExampleBackend(BaseBackend):
    def __init__(self):
        # do some setup here, e.g. processing your own settings

    ...

backend = ExampleBackend()
```

If this code is stored in a file `myproject/myapp/example.py` then the setting to use this backend would be:

```
CAMPAIGN_BACKEND = 'myproject.myapp.example'
```

Each backend must define a `backend` variable, which should be an instance of the backend.

1.4.2 Backend Methods

The following methods must be implemented by every backend:

`send_mail(self, email, fail_silently=False)`

This method takes an instance of `django.core.mail.EmailMessage` as argument and is responsible for whatever is needed to send this email to the recipient.

The `fail_silently` argument specifies whether exceptions should bubble up or should be hidden from upper layers.

1.5 Templates

1.5.1 E-Mail Templates

All e-mail templates are pure Django Templates, please see the Django Template Documentation for details. This document only contains some parts specific to django-campaign.

E-Mail Template Context

At the time the e-mail templates are rendered the following variables are available in the template context:

- `recipient` - The object which receives the email. This can be whatever ContentType is specified in the SubscriberList that is currently processed.
- `recipient_email` - The email address to which the current email is send.

If the Campaign is marked for online viewing the context will also contain the following variables:

- `view_online_url` - The URL at which the campaign can be seen online
- `viewed_online` - If the campaign is viewed with a webbrowser this variable is `True`, otherwise it is not present. This is usefull to hide the 'view online' links from if the campaign is viewed with a webbrowser.
- `site_url` - The URL of the current django Site. See django.contrib.sites for more information.

If any *CAMPAIGN_CONTEXT_PROCESSORS* are defined their results are also available in the context at the time the email is sent. The results of the CAMPAIGN_CONTEXT_PROCESSORS will not be available if the campaign is viewed online.

1.5.2 Other Templates

If you use the built-in support for handling subscriptions and unsubscriptions you most probably want to override the template `campaign/base.html` somewhere in your projects TEMPLATE_DIRS. The bundled base.html template is only a placeholder to make developing and testing easier.

Of course, the templates `campaign/subscribe.html` and `campaign/unsubscribe.html` can also be overwritten to adapt to your site. They are kept pretty simple and only demonstrate how things should work.

1.6 Philisophy behind some of the concepts of django-campaign

1.6.1 Why is there no Subscriber-Model?

I've tried to use a bunch of different Subscriber-Models bundled with the app itself but none of them was usable for more than one use-case so I decided to drop the concept of a Subscriber-Model and instead added a mechanism for you to hook your own Subscriber (or User or whatever) model into the flow.

By adding a SubscriberList Object with a pointer to the ContentType of your Model and by optionally adding lookup kwargs to narrow the selection you can specifiy which objects of your model class for a list of subscribers. You can even build SubscriberLists for different Models and send a Campaign in one step to multiple SubscriberLists.

Adding a SubscriberList for all active Users present in the django.contrib.auth module one would simply add a SubscriberList object:

```
from django.contrib.auth.models import User
from django.contrib.contenttypes.models import ContentType
from campaing.models import SubscriberList

obj = SubscriberList.objects.create(
            content_type=ContentType.objects.get_for_model(User),
            filter_condition={'is_active': True}
        )
```

Of course this can also be done using Django's built-in admin interface. Simply select Content type `user` and Filter condition `{"is_active": true}`.

Being able to add any number and combinations of ContentTypes and lookup kwargs and assining one or multiple SubscriberLists to a Campaign one should be able to map any real-world scenario to the workflow. If a subscriber is present in multiple SubscriberLists this is not a problem because the code makes sure that every Campaign is only sent once to every given email address.