django-transplant Documentation

Release 0.0.2

Karol Majta

Sep 27, 2017

Contents

1 About **3**
- 1.1 Overview . . . 3
- 1.2 Rationale behind django-transplant . . . 3

2 Installation **5**
- 2.1 Getting the code . . . 5
- 2.2 Requiremets . . . 5
- 2.3 Installing . . . 5

3 Configuration **7**
- 3.1 Configuring INSTALLED_APPS . . . 7
- 3.2 Hooking up default URLs . . . 7
- 3.3 Hooking up view in your urls.py . . . 7
- 3.4 Configuring TRANSPLANT_OPERATIONS in your settings.py . . . 8
- 3.5 Available settings . . . 9

4 django-transplant's API **11**
- 4.1 Surgery class . . . 11
- 4.2 Surgeon class . . . 12
- 4.3 Extending django-template . . . 12

5 Indices and tables **13**

Contents:

CHAPTER 1

About

Overview

Django-transplant is an app for performing easy merges of django user accounts. It should play nicely with any third party social authentication backend.

Rationale behind django-transplant

When using third party authentication apps for django it is common that users create more than one account for themselves. While some apps provide ways to easily attach social accounts so django's native user accounts, others don't. This approach is often based on email addresses pulled from the authentication services, and may sometimes fail (i.e. if no e-mail data is available).

Django-transplant enables quick merges of user accounts performed on demand. Moreover it allows you to keep your logic on how to perform these merges away from views. Django-transplant provides some basic classes to reduce your boilerplate, and allow easy extensibility.

CHAPTER 2

Installation

Getting the code

Source code is available at:

> http://github.com/lolek09/django-transplant

Requiremets

Django-transplant requires:

- Python 2.7
- Django 1.2.5

If you plan to develop, or run the test suite you should also install:

- Mock

This dependency **will not** be installed automatically via pip.

Installing

To install with pip issue:

```
pip install django-transplant
```

CHAPTER 3

Configuration

Configuring INSTALLED_APPS

Add 'transplant' to your `INSTALLED_APPS`. If you plan to run the test suite you should also add 'transplant.tests':

```
INSTALLED_APPS += (
                'transplant',
                'transplant.tests', # this is optional
        )
```

Hooking up default URLs

For your convenience django-transplant provides a default view for performing User merges. You can use it like any FormView, and it's name is `transplant_merge`. It expects a default template in 'transplant/merge.html'.

To hook it up just add it to your `urlconf` at any URL:

```
urplatterns = patterns('',
        ...
        url(r'^accounts/merge/$', include('transplant.urls')),
        ...
)
```

Hooking up view in your urls.py

`transplant.views.TransplantMergeView` is a subclass of generic `FormView` so you can hook it directly to your urls. You can pass it's arguments like you would to any other generic view:

```
...
from django.contrib.auth.decorators import login_required

from views import TransplantMergeView
...

urlpatterns = patterns('',
    ...
    url(r'^$',
        login_required(TransplantMergeView.as_view(
            template_name='custom/template/name.html')
        ),
        name='custom_name'
    ),
    ...
)
```

Configuring TRANSPLANT_OPERATIONS in your settings.py

After setting URLs yous should be able to get the merge form and submit it, but it will have no effect. To utilize default merges you must set `TRANSPLANT_OPERATIONS` in your settings.py:

```
TRANSPLANT_OPERATIONS = (
    (
        'transplant.tests.models.CustomProfile',
        'transplant.surgeons.DefaultSurgeon',
        {}
    ),
    (
        'transplant.tests.models.Item',
        'transplant.tests.surgeons.DefaultSurgeon',
        {'user_field': 'owner'}
    ),
    (
        'transplant.tests.models.Message',
        'transplant.tests.surgeons.DefaultSurgeon',
        {'manager': 'unread'}
    ),
)
```

`TRANSPLANT_OPERATIONS` consists of triples, each one of them specifies:

1. Path to model class to be merged.
2. Path to `Surgeon` class to be used during the merge.
3. Extra arguments.

Currently supported extra arguments are:

- `user_field` - name of the user field that will be used by the Surgeon during the merge (defaults to ‘user’).
- `manager` - name of Manager used during the merge. In the example above only messages accessible via the ‘unread’ manager will be merged.

You may be happy with the behavior of `DefaultSurgeon` which is:

- set field given as `'user_field'` to the user that performs the merge

- call `save()` on each entity (so that all signals are triggered)
- set the `is_active` to False on the user that is merged

If you want additional functionality consult API docs.

Available settings

Currently available settings are:

`TRANSPLANT_OPERATIONS` Allows for specification of operations to be performed during automated user merge. Widely discussed above.

`TRANSPLANT_SUCCESS_URL` Allows fot specification of URL that the user will be redirected to after successfull account merge. Defaults to `LOGIN_REDIRECT_URL`

`TRANSPLANT_FAILURE_URL` When `Debug` is set to `True` this setting takes no effect and `TransplantMergeView` will re-raise any exception. When `Debug` is set to `True` instead of raising an error, the view will redirect to provided URL. If you want it to raise error anyway set `TRANSPLANT_FAILURE_URL` to `None`. This is the default value.

CHAPTER 4

django-transplant's API

Django-transplant attempts to split logic that performs User account merges into atomic chunks that can be easily and separately maintained. `Surgery` and `Surgeon` classes perform these tasks.

Surgery class

`Surgery` class' constructor accepts two string arguments:

```
def __init__(self, model, surgeon):
    ...
```

It tries to instantiate instances of provided classes dynamically and it will raise appropriate errors if this is impossible. Django-transplant's bundled `Surgery` class accepts positional argumetn `manager` which is a string representing manager that will be provided to `Surgeon`. Example use case is:

```
my_surgery = Surgery(
  'myapp.models.Message',
  'myapp.models.DefaultSurgeon',
  manager='sent',
)
```

This will create a surgery that will grab `Message` class, get its `sent` manager and provide it to `DefaultSurgeon` instance.

`Surgeon` also provides a `merge(receiver, donor)` method that just calls `Surgeon` instance's `merge`. The `receiver` should be the instance of User that requests the merge, `donor` is the User that should be 'merged into' receiver.

In your views you will probably want to use Surgery classes like this:

```
# build a list of surgeries
surgeries = []
surgeries.append(Surgery(...))
...
```

```
# perform merge using each surgery object
for surgery in surgeries:
    surgery.merge(self.request.user, some_other_user)
```

Surgeon class

Django-transplant provides three generic `Surgeon` classes. They reside in `transplant.surgeons` module. Each of them implements a single `merge` method which takes two arguments - *receiver* and *donor* User instances. This method accepts a keyword argument `user_field` which should be used on provided model to change the field that will be updated during the merge.

`NopSurgeon` This `Surgeon` just sets up `self.manager` and `self.user_field` with an instance of `Manager` and a `string` respectively. It's merge method does nothing, but you are encouraged to subclass `NopSurgeon` if writing new `Surgeon` classes.

`DefaultSurgeon`

Subclass of `NopSurgeon`. Its merge method will:

- set `donor.is_active` to false and donor will be saved.
- get all objects from provided `Manager` and set their field provided by 'user_field' to `receiver`.
- will call save on all objects from manager, so that all signals are triggered.

`BatchSurgeon` Works exactly like `DefaultSurgeon` but won't call save methods. No signals will be triggered.

Extending django-template

Writing new subclasses of `Surgeon` and `Surgery` is easy.

While subclassing or writing new `Surgery` classes pleas follow the convention that `__init__` takes positional argument `manager` that is provided later on to `Surgeon` to keep consistennt with django-transplant's core.

While subclassing `Surgeon` classes override `merge` following the convention to accept `user_field`.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search