staff Documentation

Release 1.2

me

February 06, 2016

Contents


EMPLOYEE
ID
123-456-789

CHAPTER 1

Goals

In the Django Authentication package all users use the same model/profile. This can be a drawback if you have lots of users and you want different information stored for your staff members, such as bio, contact info, etc. It is also handy for linking to models for author fields so the lookup is quicker.

The staff models are automatically created and managed by the `is_staff` property of `Users`. The staff profile is automatically added to the `User` admin screen.

CHAPTER 2

Contents

2.1 Getting Started

2.1.1 Installation

1. `pip install django-staff`
2. Add to `INSTALLED_APPS` in `settings.py`

2.1.2 How it works

1. Django Staff modifies (monkey patches) Django's built-in `User` model's admin, adding an inline form to the `Staff` model.
2. Django Staff registers a listener to updates to the `User` model.
3. Three possible actions can happen when a `User` is created or changed:
   (a) A staff object is created, linked to the user object and permission is added for all available sites.
   (b) The existing staff object is updated, keeping the first name, last name and e-mail in sync and also confirming the active status of the staff object.
   (c) The existing staff object is marked as inactive

2.1.3 Using Django Staff

1. Create a new `User`
2. Make sure their `staff status` is checked
3. Click the `Save and continue editing` button.
4. When the page reloads, you'll notice the additional fields at the bottom of the page.

2.2 Customizing StaffMember

While the *StaffMember* model is meant to be general, sometimes you need something extra. You can create your own subclass of *StaffMember* for tweaking. In the example project you can browse the `mystaff` app. To test it out:

1. Comment out `'staff',` from `INSTALLED_APPS` in `settings.py`
2. Uncomment `'mystaff',` from `INSTALLED_APPS` in `settings.py`
3. Delete the `dev.db` file
4. Run `./manage.py syncdb`
5. Create a new superuser when prompted

2.2.1 Create the custom class

Your custom *StaffMember* model is going to subclass `BaseStaffMember`, which is an abstract version of `StaffMember`. Add your additional fields to the class.

```
from django.db import models
from django.db.models.signals import post_save
from django.contrib.auth.models import User

from staff.models import BaseStaffMember, get_staff_updater

class MyStaffMember(BaseStaffMember):
    github = models.CharField(max_length=50, blank=True)


update_staff_member = get_staff_updater(MyStaffMember)
post_save.connect(update_staff_member, sender=User)
```

2.2.2 Connect the signal

You need to manually connect the `post_save` signal to a function that keeps your custom staff member class in sync.

```
from django.db import models
from django.db.models.signals import post_save
from django.contrib.auth.models import User

from staff.models import BaseStaffMember, get_staff_updater

class MyStaffMember(BaseStaffMember):
    github = models.CharField(max_length=50, blank=True)

```

```
10
11 update_staff_member = get_staff_updater(MyStaffMember)
12 post_save.connect(update_staff_member, sender=User)
```

1. Import `get_staff_updater()` from `staff.models`. See line 5 in the example.
2. Execute it, passing in your model, and assign it to a variable. See line 11 in the example.
3. Import `post_save` from `django.db.models.signals`. See line 2 in the example.
4. Finally connect the `post_save` signal to your staff updater variable as in line 12 in the example.

2.2.3 Create your admin class

The admin class is more complicated. It consists of three parts: customizing the `StaffMemberAdmin` class, creating a custom `UserAdmin`, and finally swapping out the currently registered `UserAdmin` class with yours.

Your own admin class

Your admin class simply needs to redefine the fieldsets and model of the `StaffMemberAdmin` class.

```
1  from django.contrib.auth.models import User
2  from django.contrib.auth.admin import UserAdmin
3  from django.contrib import admin
4  from staff.admin import StaffMemberAdmin
5
6  from .models import MyStaffMember
7
8
9  class MyStaffMemberAdmin(StaffMemberAdmin):
10     fieldsets = (
11         ('Personal Info', {'fields': ('bio', 'photo', 'website', 'phone',)}),
12         ('Social Media', {'fields': ('github', 'twitter', 'facebook', 'google_plus')}),
13         ('Responsibilities', {'fields': ('sites',)}),
14     )
15     model = MyStaffMember
16
17 class MyStaffUserAdmin(UserAdmin):
18     """
19     Subclasses the UserAdmin to add the staffmember as an inline.
20     """
21     inlines = [MyStaffMemberAdmin, ]
22
23 admin.site.unregister(User)
24 admin.site.register(User, MyStaffUserAdmin)
```

The class is very straightforward. Since we only added one field, `github`, we copy the `fieldsets` value from the base class and add that field in.

Then we set the model to our new model.

Making a custom User admin class

We need to add an inline class to the current `UserAdmin`.

```
from django.contrib.auth.models import User
from django.contrib.auth.admin import UserAdmin
from django.contrib import admin
from staff.admin import StaffMemberAdmin

from .models import MyStaffMember


class MyStaffMemberAdmin(StaffMemberAdmin):
    fieldsets = (
        ('Personal Info', {'fields': ('bio', 'photo', 'website', 'phone',)}),
        ('Social Media', {'fields': ('github', 'twitter', 'facebook', 'google_plus')}),
        ('Responsibilities', {'fields': ('sites',)}),
    )
    model = MyStaffMember

class MyStaffUserAdmin(UserAdmin):
    """
    Subclasses the UserAdmin to add the staffmember as an inline.
    """
    inlines = [MyStaffMemberAdmin, ]

admin.site.unregister(User)
admin.site.register(User, MyStaffUserAdmin)
```

This is merely sublassing the existing `UserAdmin` and adding our own `inlines` attribute equal to a list containing the new admin class defined above.

Re-registering the UserAdmin

Now we carefully swap the old `UserAdmin` with our `UserAdmin` subclass.

```
from django.contrib.auth.models import User
from django.contrib.auth.admin import UserAdmin
from django.contrib import admin
from staff.admin import StaffMemberAdmin

from .models import MyStaffMember


class MyStaffMemberAdmin(StaffMemberAdmin):
    fieldsets = (
        ('Personal Info', {'fields': ('bio', 'photo', 'website', 'phone',)}),
        ('Social Media', {'fields': ('github', 'twitter', 'facebook', 'google_plus')}),
        ('Responsibilities', {'fields': ('sites',)}),
    )
    model = MyStaffMember

class MyStaffUserAdmin(UserAdmin):
    """
    Subclasses the UserAdmin to add the staffmember as an inline.
    """
    inlines = [MyStaffMemberAdmin, ]

admin.site.unregister(User)
admin.site.register(User, MyStaffUserAdmin)
```

Django's admin has the ability to both register an admin class and unregister an admin class. After removing any admin classes associated with the `User` class, we register and associate our custom user admin class.

2.2.4 Gather the templates

Django staff includes a set of templates for various Django versions. Since we'll remove `'staff'` from `INSTALLED_APPS`, Django won't find them any more. We need to copy them into either the project's templates directory or your application's template directory.

The templates, named `staff.html` and `staff13.html`, need to go into:

```
templates
   admin
      edit_inline
         staff.html
         staff13.html
```

2.2.5 Remove staff from INSTALLED_APPS

If Django Staff is still included in your `INSTALLED_APPS` setting, you'll have a bit of redundancy. Make sure that `'staff'` is not in that list. It still must remain available to your new application, so don't don't uninstall the library.

2.3 Reference

2.3.1 Settings

Default Settings

```
DEFAULT_SETTINGS = {
    'PHOTO_STORAGE': settings.DEFAULT_FILE_STORAGE,
    'ORDERING': ('last_name', 'first_name'),
}
```

PHOTO_STORAGE

Default: `DEFAULT_FILE_STORAGE`

How you wish to store photos of staff members

ORDERING

Default: `('last_name', 'first_name')`

How the staff members are ordered in lists by default.

2.3.2 Staff Model Reference

StaffMember

class `StaffMember`

`user`
Required `ForeignKey User`

The `User` to which this profile relates. Choices are limited to active users.

`first_name`
`CharField(150)`

The first name of the user. It is automatically synced with the user object and is provided in this model for convenience.

`last_name`
`CharField(150)`

The last name of the user. It is automatically synced with the user object and is provided in this model for convenience.

`slug`
Required `SlugField`

A URL-friendly version of the first and last name.

`email`
`EmailField`

The email address of the user. It is automatically synced with the user object and is provided in this model for convenience.

`bio`
`TextField`

An in-depth, riveting account of the user's life.

`is_active`
Required `BooleanField` *Default:* `True`

`True` indicates a current staff member. `False` indicates a former staff member.

`phone`
`PhoneNumberField`

A series of digits which, when typed into telephonic devices, might establish a vocal connection to the user.

`photo`
`RemovableImageField`

A visual, digital representation of the user. The image is stored based on the *PHOTO_STORAGE* setting.

`photo_height`
`IntegerField`

An automatically-managed reference to the height of the uploaded photo.

`photo_width`
`IntegerField`

An automatically-managed reference to the width of the uploaded photo.

twitter
`CharField(100)`

The staff member's Twitter ID

facebook
`CharField(100)`

The staff member's Facebook ID

google_plus
`CharField(100)`

The staff member's Google Plus account

website
`URLField`

A custom web site for the staff member.

sites
`ManyToManyField Site`

The sites that the staff member works on.

get_full_name()
A convenient way to concatenate the first and last name.

Returns *unicode*

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

B

E

F

G

I

L

P

S

T

U

W