django-generic-confirmation Documentation

Release 0.1

Arne Brodowski

Jun 07, 2017

Contents

Django-generic-confirmation makes it easy for developers to add forms to a webapplication where the submitted data should only be used after an out-of-band confirmation was done. For example if a user wants to change his email address, generic-confirmation will make it really easy for the developer to add an out-of-band confirmation process (sending an email with a random link to the user) before saving the new email address to the database.

The core of django-generic-confirmation is fully unit-tested and the app is in use at a few real-world projects confirming email addresses and mobile phone numbers. Feel free to read about the *usage*.

CHAPTER 1

Contents

Using django-generic-confirmation

django-generic-confirmation tries to solve a few real use-cases I stumpled upon in different projects in a generic way.

Use-cases

Use-cases include but are not limited to:

- New user signs up and the account should only be activated after the email address is confirmed by clicking a link in an email sent to the user.
- An already registered user wants to change his email address. Bevor doing the change, the user gets an email and has to click a link to confirm the change.
- A user enters his mobile phone numer. To confirm, that the user owns the number, a short random code is sent via sms and the user has to enter the code into a form to confirm the number.
- A postal address should be confirmed by sending the user a letter with a short random code, the user receives the letter and has to enter the code into a form to confirm his address.
- Same as above for bank accounts. You send the user a small amount of money together with a short code.

django-generic-confirmation will solve this use-cases and make it easy for you to integrate this and any other things into your project with minimal effort.

Based on these use-cases a pattern of deferred modelform saving emerged.

Inheriting from DeferredForm you get a ModelForm which will not modify or create the object on save() but will store everything needed to finish the action in the database identified by a random token. Later, whenever this token is received by the application (and if it's not expired), the form will be saved as if the data was received just now instead of the token.

One caveat: If a form is defered which involves ForeignKey oder ManyToMany Fields and the related objects are deleted while the form is deferred, than an error will be raised on confirmation.

For the developer the DeferredForm works like a normal ModelForm and all you have to do to is write your view as if it should change the data right now, under the hood it is not executed until it is confirmed.

I created this project because I think too many apps exist to solve a very similar problem. Take a look at:

- django-registration: Creates User objects and activates them if the email is confirmed by visiting a random link sent to the email address.
- django-email-confirmation: Based on django-registration, but used to manage the case, where a user has more than one email address and every new address has to be confirmed.
- django-confirmation: based on django-registration and django-email-registration, exists to confirm object-creation via email, much like django-registration but a bit more flexible because any django model can be used. Marks objects as confirmed by changing a field on the object.

API example

This is the code you have to write to turn some normal editing workflow into a edit-and-confirm workflow.

```
def my_view(request):
    """
    example view to change the user's email address
    """
    if request.method == 'POST':
        form = EmailChangeForm(request.POST, instance=request.user)
        if form.is_valid():
            form.save()
            # redirect user and display a message explaining how to
            # confirm the change
            # request.user.message_set.create(message=u"...")
            # return HttpResponseRedirect(...)
            # _or_
            # send user to a page which explains the confirmation process
            return render_to_response(...)
    else:
        form = EmailChangeForm()
    return render_to_response(...)
```

the magic happens in the form, which should be inherited from `generic_confirmation.forms.DeferredForm`.

```
class EmailChangeForm(DeferredForm):
    class Meta:
        model = User
        fields = ('email',)
```

your url-conf:

```
(r'^confirm/', include('generic_confirmation.urls'))
```

Notification

The one part left is how to get the random token to the user who has to confirm the change. This does not neccessarily has to be the user who made the change in the first step. This could also be some site-moderator for example.

Notification of someone about some action which should be confirmed can happen via different channels, most notably email and sms. django-notification would be great for this, but the pluggable-backend branch is not finished yet (isn't it?) ...

Signal-based notification

To make a long story short, currently only the signal `confirmation_required` is fired and it's your task to listen for it and take the appropriate action.

```
from generic_confirmation import signals

def send_notification(sender, instance, **kwargs):
        """ a signal receiver which does some tests """
        print sender # the class which is edited
        print instance # the DeferredAction
        print instance.token # needed to confirm the action


signals.confirmation_required.connect(send_notification)
```

If you pass `user=request.user` to the form.save()-method, then the signal will provide a user argument pointing to the user who requested the change. It's also possible to pass another user to save method to inform an admin or so.

```
def send_notification(sender, instance, user, **kwargs):
    body = render_to_string('confirmation_mail.html', {'user': user, 'token':␣
↪instance.token})
    send_mail("subject", body, recipient_list=[user.email,])
    # _or_
    # send_mail("subject", body, recipient_list)

signals.confirmation_required.connect(send_notification)
```

Form-based notification

The second way is to provide a method named `send_notification` on the form-class itself. The method takes the request.user (if given) and the instance which should be confirmed as arguments.

```
class EmailChangeForm(DeferredForm):
    def send_notification(self, user=None, instance=None):
        send_mail("please confirm your new address",
            render_to_string("confirm_mail.txt", {'token': instance.token, 'user':␣
↪user}),
            recipient_list=[self.cleaned_data['email'],])

    class Meta:
        model = User
        fields = ('email',)



class PhoneNumberChangeForm(DeferredForm):
    def send_notification(self, user=None, instance=None):
        send_sms({'to': self.cleaned_data['mobile_number'],
                'text': render_to_string('confirm_phone.txt',
                        {'token': instance.token, 'user': user})
                })
```

```
    class Meta:
        model = UserProfile
        fields = ('mobile_number',)
```

Some of the default notification methods will be provided as mixin classes soon.