

Authentification Django dans MDM

Author : PimenTech
Contact : root@_NOSPAM_pimentech.net
Revision : 1.81
Date : 2007-06-28
Tags : django authentification mdm

L'authentification du nouvel intranet Century 21 a été complètement réécrite pour bénéficier des mécanismes avancés fournis par Django.

Backends d'authentification

Depuis le 28 juin 2006, la branche `multi-auth` de Django a été intégrée dans la branche principale. Cette branche ajoute la possibilité de définir des **backends d'authentification** supplémentaires, qui permettent de s'authentifier autrement que par la méthode standard (en utilisant `User`).

En Django, un backend d'authentification est simplement une classe Python qui définit deux méthodes :

- une méthode `authenticate()` qui prend en paramètre un *login* et un *mot de passe*, et qui retourne un objet de la classe `User` Django.
- une méthode `get_user()`, qui à partir d'un *user_id*, retourne l'objet `User` en question.

Ces backends sont déclarés dans `settings.py` de la manière suivante :

```
AUTHENTICATION_BACKENDS = (  
    'dj.auth.backends.C21Backend',  
    'django.contrib.auth.backends.ModelBackend',  
)
```

La variable `AUTHENTICATION_BACKENDS` est un tuple de chemins vers des backends. Ces backends sont « essayés » dans l'ordre, les un après les autres. Dès que l'on peut s'authentifier auprès de l'un d'eux, l'utilisateur est accepté.

On a donc défini un backend supplémentaire, `dj.auth.backends.C21Backend`, qui authentifie l'utilisateur en utilisant `Est_employe_par` :

```
from django.contrib.auth.models import User  
  
from dj.auth.models import C21UserProfile  
from dj.mdm.models import EstEmployePar  
  
class C21Backend:  
    """  
    Authenticate against dj.mdm.models.EstEmployePar  
    """  
    def authenticate(self, username=None, password=None):  
        try:  
            eep = EstEmployePar.objects.filter(username=username, pas  
            agence = eep.agence  
            personnel = eep.personnel  
            try:  
                user = User.objects.get(username=username)  
            except User.DoesNotExist:
```

```
        user = User(username=username, password='blank')
        user.first_name = personnel.prenom
        user.last_name = personnel.nom
        user.is_staff = eep.gestion_intranet
        user.save()
        profile = C21UserProfile(user=user)
        profile.est_employe_par = eep
        profile.agence = agence
        profile.personnel = personnel
        profile.save()

    return user
except EstEmployePar.DoesNotExist:
    return None

def get_user(self, user_id):
    try:
        return User.objects.get(pk=user_id)
    except User.DoesNotExist:
        return None
```

Ce backend renseigne automatiquement les champs de l'objet user avec les champs de est_employe_par les plus utiles (nom, prenom, gestion_intranet, etc.). Ces valeurs sont mises à disposition des vues et des templates Django. De plus, on a accès aux lignes des tables est_employe_par, agence et personnel associées à l'utilisateur authentifié, grâce au système de [profil](#) mis en place par Django.

Si l'authentification C21 échoue, le système essaiera automatiquement l'authentification Django standard de façon transparente. Ainsi, on peut s'authentifier en utilisant un User Django qui n'existe pas dans Est_employe_par (très utile pour le webmaster par exemple).

De plus, comme l'authentification est le mécanisme d'authentification standard de Django, un utilisateur qui a les droits suffisants (is_superuser à True) est automatiquement reconnu par l'admin Django.

Le système de profil Django

Comme on a souvent besoin d'accéder à des informations relatives à l'utilisateur authentifié qui ne sont pas dans user, Django a mis en place un système de **profil**.

Ce système s'utilise de la façon suivante. Dans settings.py, on ajoute la ligne suivante :

```
AUTH_PROFILE_MODULE = 'auth.C21UserProfile'
```

où auth est ici le nom de l'application (et non pas le chemin complet du modèle ! dj.auth.models.C21UserProfile ne fonctionnera pas), et C21UserProfile le nom du modèle associé à l'utilisateur. Ce modèle peut contenir n'importe quel champ, mais il doit impérativement posséder un champ user qui correspond à une référence vers la table User de Django.

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

from dj.mdm.models import EstEmployePar, Agence, Personnel

class C21UserProfile(models.Model):
    est_employe_par = models.ForeignKey(EstEmployePar)
    agence = models.ForeignKey(Agence)
    personnel = models.ForeignKey(Personnel)
    user = models.OneToOneField(User, core=True)
```

Lors de l'authentification de l'utilisateur, la jointure avec le profil est faite automatiquement, et on peut y accéder à partir du user en faisant user.get_profile(). Par exemple pour récupérer le code agence de l'utilisateur authentifié, on écrira :

```
request.user.get_profile().agence.code
```

dans une vue, ou :

```
{{ user.get_profile.agence.code }}
```

dans un template. Voir plus bas pour l'accès aux informations de l'[utilisateur](#) dans les templates.

Gestion des URL

```
from django.conf.urls.defaults import *

urlpatterns = patterns('',
    ...
    (r'^accounts/login/$', 'django.contrib.auth.views.login', { 'template_name':
    (r'^accounts/logout/$', 'django.contrib.auth.views.logout', { 'template_name'
    ...
)
```

Comme ce n'est pas trop conseillé, Django ne prévoit pas la possibilité de passer le login et le mot de passe en GET dans l'URL. Mais nous avons temporairement besoin de cette possibilité pour pouvoir rajouter des liens vers le nouvel Intranet à partir de l'ancien, sans avoir à s'authentifier à nouveau.

Nous avons donc rajouté la vue suivante dans `dj.auth.views` :

```
from django import forms
from django.shortcuts import render_to_response
from django.template import RequestContext
from django.contrib.sites.models import Site
from django.http import HttpResponseRedirect
from django.contrib.auth import REDIRECT_FIELD_NAME

from dj.auth.forms import AuthenticationForm

def login(request, template_name='registration/login.html'):
    "Displays the login form and handles the login action."
    manipulator = AuthenticationForm(request)
    redirect_to = request.REQUEST.get(REDIRECT_FIELD_NAME, '')
    if request.GET.has_key('username'):
        request.POST = request.POST.copy()
        request.POST.update({ 'username': request.GET['username'], 'password': request.GET['password'] })
    if request.POST:
        errors = manipulator.get_validation_errors(request.POST)
        if not errors:
            # Light security check -- make sure redirect_to isn't garbage
            if not redirect_to or '://' in redirect_to or ' ' in redirect_to:
                redirect_to = '/accounts/profile/'
            from django.contrib.auth import login
            login(request, manipulator.get_user())
            #request.session.delete_test_cookie()
            return HttpResponseRedirect(redirect_to)
    else:
        errors = {}
    #request.session.set_test_cookie()
    return render_to_response(template_name, {
        'form': forms.FormWrapper(manipulator, request.POST, errors),
        REDIRECT_FIELD_NAME: redirect_to,
        'site_name': Site.objects.get_current().name,
    }, context_instance=RequestContext(request))
```

Cette vue regarde le login a été passé en paramètre GET, et rajoute les valeurs correspondantes dans `request.POST`, avant de propager l'appel à la fonction de login fournie par Django. Dans `urls.py`, il suffit donc de remplacer la ligne de login par :

```
(r'^accounts/login/$', 'dj.auth.views.login', { 'template_name': 'auth/login.html
```

pour gérer de façon transparente le login dans l'URL.

Par exemple, pour se connecter directement sur l'application coups de coeur, on rajoutera un lien de la forme :

```
http://dj.pimentech.net/accounts/login/?username=toto&password=1234&next=/cdc/
```

TODO

- login en dur /accounts/login/
- logout peut changer
- redirection next
- templates login/logout

Les décorateurs d'authentification

Le module d'authentification Django fournit deux **décorateurs** qui permettent de contrôler l'accès aux vues : `login_required()` et `user_passes_test()`. Par exemple, pour limiter l'accès à une vue aux seuls utilisateurs authentifiés, on écrit simplement le code suivant :

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

def my_view(request):
    ...
    my_view = login_required(my_view)
```

Lors de l'accès à la vue, l'utilisateur sera automatiquement redirigé vers la page de login si il n'est pas authentifié, puis à nouveau redirigé vers la vue.

À partir de Python 2.4, on peut écrire plus simplement :

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

@login_required
def my_view(request):
    ...
```

Pour faciliter l'ajout de nouvelles vues, deux décorateurs spécifiques à l'authentification MDM sont fournis :

- `admin_required()`, qui vérifie que l'utilisateur a les droits administrateurs (pour les applications *webmaster*, comme la gestion des cuts par exemple)
- `gestion_intranet_required()`, qui vérifie que l'utilisateur a les droits *manager* sur l'intranet C21 (pour l'application des offres d'emploi par exemple).

Ces deux décorateurs sont définis comme suit (dans `dj.auth.decorators`) :

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

admin_required = user_passes_test(lambda u: u.is_authenticated() and u.is_superuser)
gestion_intranet_required = user_passes_test(lambda u: u.is_authenticated() and u
```

On peut donc écrire :

```
from dj.auth.decorators import gestion_intranet_required

def liste_offres(request, agence=None):
    ...
    liste_offres = gestion_intranet_required(liste_offres)
```

Accès aux données de l'utilisateur dans les templates

Accéder directement à `request` dans les templates est une très mauvaise pratique Django. Pour éviter cela, Django définit des **context processors**. Lorsqu'on appelle `render_to_response`, il faut écrire :

```
return render_to_response('app/template.html',  
                          { ... },  
                          context_instance=RequestContext(request))
```

et le contexte est automatiquement enrichi des valeurs fournies par les différents context processors. Dans `settings.py`, on écrit :

```
TEMPLATE_CONTEXT_PROCESSORS = (  
    'django.core.context_processors.auth',  
    ...  
)
```

pour pouvoir accéder directement à `user` (entres autres) dans les templates.

TODO

- cache
- plus tard : permissions ?
- model subclassing
- autre : modèle mdm dans admin