django-user-accounts Documentation

Release 2.0.3

Pinax

Jun 09, 2017

Contents

Provides user accounts to a Django project.

CHAPTER 1

Development

The source repository can be found at https://github.com/pinax/django-user-accounts/

Contents

Installation

Install the development version:

```
pip install django-user-accounts
```

Add `account` to your `INSTALLED_APPS` setting:

```
INSTALLED_APPS = (
    # ...
    "account",
    # ...
)
```

See the list of *Settings* to modify the default behavior of django-user-accounts and make adjustments for your website.

Add `account.urls` to your URLs definition:

```
urlpatterns = patterns("",
    ...
    url(r"^account/", include("account.urls")),
    ...
)
```

Add `account.context_processors.account` to `TEMPLATE_CONTEXT_PROCESSORS`:

```
TEMPLATE_CONTEXT_PROCESSORS = [
    ...
```

```
    "account.context_processors.account",
    ...
]
```

Add `account.middleware.LocaleMiddleware` and `account.middleware.TimezoneMiddleware` to `MIDDLEWARE_CLASSES`:

```
MIDDLEWARE_CLASSES = [
    ...
    "account.middleware.LocaleMiddleware",
    "account.middleware.TimezoneMiddleware",
    ...
]
```

Optionally include `account.middleware.ExpiredPasswordMiddleware` in `MIDDLEWARE_CLASSES` if you need password expiration support:

```
MIDDLEWARE_CLASSES = [
    ...
    "account.middleware.ExpiredPasswordMiddleware",
    ...
]
```

Once everything is in place make sure you run `migrate` to modify the database with the `account` app models.

Dependencies

`django.contrib.auth`

This is bundled with Django. It is enabled by default with all new Django projects, but if you adding django-user-accounts to an existing project you need to make sure `django.contrib.auth` is installed.

`django.contrib.sites`

This is bundled with Django. It is enabled by default with all new Django projects. It is used to provide links back to the site in emails or various places in templates that need an absolute URL.

django-appconf

We use django-appconf for app settings. It is listed in `install_requires` and will be installed when pip installs.

pytz

pytz is used for handling timezones for accounts. This dependency is critical due to its extensive dataset for timezones.

Usage

This document covers the usage of django-user-accounts. It assumes you've read *Installation*.

django-user-accounts has very good default behavior when handling user accounts. It has been designed to be customizable in many aspects. By default this app will:

- enable username authentication
- provide default views and forms for sign up, log in, password reset and account management
- handle log out with POST
- require unique email addresses globally
- require email verification for performing password resets

The rest of this document will cover how you can tweak the default behavior of django-user-accounts.

Limiting access to views

To limit view access to logged in users, normally you would use the Django decorator `django.contrib.auth.decorators.login_required`. Instead you should use `account.decorators.login_required`.

Customizing the sign up process

In many cases you need to tweak the sign up process to do some domain specific tasks. Perhaps you need to update a profile for the new user or something else. The built-in `SignupView` has hooks to enable just about any sort of customization during sign up. Here's an example of a custom `SignupView` defined in your project:

```
import account.views


class SignupView(account.views.SignupView):

    def after_signup(self, form):
        self.update_profile(form)
        super(SignupView, self).after_signup(form)

    def update_profile(self, form):
        profile = self.created_user.profile  # replace with your reverse one-to-one
↪profile attribute
        profile.some_attr = "some value"
        profile.save()
```

This example assumes you had a receiver hooked up to the *post_save* signal for the sender, *User* like so:

```
from django.dispatch import receiver
from django.db.models.signals import post_save

from django.contrib.auth.models import User

from mysite.profiles.models import UserProfile


@receiver(post_save, sender=User)
def handle_user_save(sender, instance, created, **kwargs):
    if created:
        UserProfile.objects.create(user=instance)
```

You can define your own form class to add fields to the sign up process:

```
# forms.py

from django import forms
```

```
from django.forms.extras.widgets import SelectDateWidget

import account.forms


class SignupForm(account.forms.SignupForm):

    birthdate = forms.DateField(widget=SelectDateWidget(years=range(1910, 1991)))

# views.py

import account.views

import myproject.forms


class SignupView(account.views.SignupView):

   form_class = myproject.forms.SignupForm

   def after_signup(self, form):
       self.create_profile(form)
       super(SignupView, self).after_signup(form)

   def create_profile(self, form):
       profile = self.created_user.profile  # replace with your reverse one-to-one␣
↪profile attribute
       profile.birthdate = form.cleaned_data["birthdate"]
       profile.save()
```

To hook this up for your project you need to override the URL for sign up:

```
from django.conf.urls import patterns, include, url

import myproject.views


urlpatterns = patterns("",
    url(r"^account/signup/$", myproject.views.SignupView.as_view(), name="account_
↪signup"),
    url(r"^account/", include("account.urls")),
)
```

Note: Make sure your `url` for `/account/signup/` comes *before* the `include` of `account.urls`. Django will short-circuit on yours.

Using email address for authentication

django-user-accounts allows you to use email addresses for authentication instead of usernames. You still have the option to continue using usernames or get rid of them entirely.

To enable email authentication do the following:

1. check your settings for the following values:

```
ACCOUNT_EMAIL_UNIQUE = True
ACCOUNT_EMAIL_CONFIRMATION_REQUIRED = True
```

Note: If you need to change the value of `ACCOUNT_EMAIL_UNIQUE` make sure your database schema is modified to support a unique email column in `account_emailaddress`.

`ACCOUNT_EMAIL_CONFIRMATION_REQUIRED` is optional, but highly recommended to be `True`.

2. define your own `LoginView` in your project:

```
import account.forms
import account.views


class LoginView(account.views.LoginView):

    form_class = account.forms.LoginEmailForm
```

3. ensure `"account.auth_backends.EmailAuthenticationBackend"` is in `AUTHENTICATION_BACKENDS`

If you want to get rid of username you'll need to do some extra work:

1. define your own `SignupForm` and `SignupView` in your project:

```
# forms.py

import account.forms


class SignupForm(account.forms.SignupForm):

    def __init__(self, *args, **kwargs):
        super(SignupForm, self).__init__(*args, **kwargs)
        del self.fields["username"]

# views.py

import account.views

import myproject.forms


class SignupView(account.views.SignupView):

    form_class = myproject.forms.SignupForm
    identifier_field = 'email'

    def generate_username(self, form):
        # do something to generate a unique username (required by the
        # Django User model, unfortunately)
        username = "<magic>"
        return username
```

2. many places will rely on a username for a User instance. django-user-accounts provides a mechanism to add a level of indirection when representing the user in the user interface. Keep in mind not everything you include in your project will do what you expect when removing usernames entirely.

Set ACCOUNT_USER_DISPLAY in settings to a callable suitable for your site:

```
ACCOUNT_USER_DISPLAY = lambda user: user.email
```

Your Python code can use user_display to handle user representation:

```
from account.utils import user_display
user_display(user)
```

Your templates can use {% user_display request.user %}:

```
{% load account_tags %}
{% user_display request.user %}
```

Allow non-unique email addresses

If your site requires that you support non-unique email addresses globally you can tweak the behavior to allow this.

Set ACCOUNT_EMAIL_UNIQUE to False. If you have already setup the tables for django-user-accounts you will need to migrate the account_emailaddress table:

```
ALTER TABLE "account_emailaddress" ADD CONSTRAINT "account_emailaddress_user_id_email_
↪key" UNIQUE ("user_id", "email");
ALTER TABLE "account_emailaddress" DROP CONSTRAINT "account_emailaddress_email_key";
```

ACCOUNT_EMAIL_UNIQUE = False will allow duplicate email addresses per user, but not across users.

Including accounts in fixtures

If you want to include account_account in your fixture, you may notice that when you load that fixture there is a conflict because django-user-accounts defaults to creating a new account for each new user.

Example:

```
IntegrityError: Problem installing fixture \
      ...'/app/fixtures/some_users_and_accounts.json': \
      Could not load account.Account(pk=1): duplicate key value violates unique␣
↪constraint \
      "account_account_user_id_key"
DETAIL:  Key (user_id)=(1) already exists.
```

To prevent this from happening, subclass DiscoverRunner and in setup_test_environment set CREATE_ON_SAVE to False. For example in a file called lib/tests.py:

```
from django.test.runner import DiscoverRunner
from account.conf import AccountAppConf

class MyTestDiscoverRunner(DiscoverRunner):

    def setup_test_environment(self, **kwargs):
        super(MyTestDiscoverRunner, self).setup_test_environment(**kwargs)
        aac = AccountAppConf()
        aac.CREATE_ON_SAVE = False
```

And in your settings:

```
TEST_RUNNER = "lib.tests.MyTestDiscoverRunner"
```

Enabling password expiration

Password expiration is disabled by default. In order to enable password expiration you must add entries to your settings file:

```
ACCOUNT_PASSWORD_EXPIRY = 60*60*24*5  # seconds until pw expires, this example shows␣
↪five days
ACCOUNT_PASSWORD_USE_HISTORY = True
```

and include *ExpiredPasswordMiddleware* with your middleware settings:

```
MIDDLEWARE_CLASSES = {
    ...
    "account.middleware.ExpiredPasswordMiddleware",
}
```

`ACCOUNT_PASSWORD_EXPIRY` indicates the duration a password will stay valid. After that period the password must be reset in order to view any page. If `ACCOUNT_PASSWORD_EXPIRY` is zero (0) then passwords never expire.

If `ACCOUNT_PASSWORD_USE_HISTORY` is False, no history will be generated and password expiration WILL NOT be checked.

If `ACCOUNT_PASSWORD_USE_HISTORY` is True, a password history entry is created each time the user changes their password. This entry links the user with their most recent (encrypted) password and a timestamp. Unless deleted manually, PasswordHistory items are saved forever, allowing password history checking for new passwords.

For an authenticated user, `ExpiredPasswordMiddleware` prevents retrieving or posting to any page except the password change page and log out page when the user password is expired. However, if the user is "staff" (can access the Django admin site), the password check is skipped.

Settings

ACCOUNT_OPEN_SIGNUP

Default: `True`

ACCOUNT_LOGIN_URL

Default: `"account_login"`

ACCOUNT_SIGNUP_REDIRECT_URL

Default: `"/"`

ACCOUNT_LOGIN_REDIRECT_URL

Default: `"/"`

ACCOUNT_LOGOUT_REDIRECT_URL

Default: `"/"`

ACCOUNT_PASSWORD_CHANGE_REDIRECT_URL

Default: `"account_password"`

ACCOUNT_PASSWORD_RESET_REDIRECT_URL

Default: `"account_login"`

ACCOUNT_PASSWORD_EXPIRY

Default: `0`

ACCOUNT_PASSWORD_USE_HISTORY

Default: `False`

ACCOUNT_REMEMBER_ME_EXPIRY

Default: `60 * 60 * 24 * 365 * 10`

ACCOUNT_USER_DISPLAY

Default: `lambda user:  user.username`

ACCOUNT_CREATE_ON_SAVE

Default: `True`

ACCOUNT_EMAIL_UNIQUE

Default: `True`

ACCOUNT_EMAIL_CONFIRMATION_REQUIRED

Default: `False`

ACCOUNT_EMAIL_CONFIRMATION_EMAIL

Default: `True`

ACCOUNT_EMAIL_CONFIRMATION_EXPIRE_DAYS

Default: `3`

ACCOUNT_EMAIL_CONFIRMATION_ANONYMOUS_REDIRECT_URL

Default: `"account_login"`

ACCOUNT_EMAIL_CONFIRMATION_AUTHENTICATED_REDIRECT_URL

Default: `None`

ACCOUNT_EMAIL_CONFIRMATION_URL

Default: `"account_confirm_email"`

ACCOUNT_SETTINGS_REDIRECT_URL

Default: `"account_settings"`

ACCOUNT_NOTIFY_ON_PASSWORD_CHANGE

Default: `True`

ACCOUNT_DELETION_MARK_CALLBACK

Default: `"account.callbacks.account_delete_mark"`

ACCOUNT_DELETION_EXPUNGE_CALLBACK

Default: `"account.callbacks.account_delete_expunge"`

ACCOUNT_DELETION_EXPUNGE_HOURS

Default: `48`

ACCOUNT_HOOKSET

Default: `"account.hooks.AccountDefaultHookSet"`

This setting allows you define your own hooks for specific functionality that django-user-accounts exposes. Point this to a class using a string and you can override the following methods:

- `send_invitation_email(to, ctx)`
- `send_confirmation_email(to, ctx)`
- `send_password_change_email(to, ctx)`
- `send_password_reset_email(to, ctx)`

ACCOUNT_TIMEZONES

Default: `list(zip(pytz.all_timezones, pytz.all_timezones))`

ACCOUNT_LANGUAGES

See full list in: https://github.com/pinax/django-user-accounts/blob/master/account/language_list.py

Templates

This document covers the implementation of django-user-accounts within Django templates. The pinax-theme-bootstrap package provides a good starting point to build from. Note, this document assumes you have read the installation docs.

Template Files

By default, django-user-accounts expects the following templates. If you don't use `pinax-theme-bootstrap`, then you will have to create these templates yourself.

Login/Registration/Signup Templates:

```
account/login.html
account/logout.html
account/signup.html
account/signup_closed.html
```

Email Confirmation Templates:

```
account/email_confirm.html
account/email_confirmation_sent.html
account/email_confirmed.html
```

Password Management Templates:

```
account/password_change.html
account/password_reset.html
account/password_reset_sent.html
account/password_reset_token.html
account/password_reset_token_fail.html
```

Account Settings:

```
account/settings.html
```

Emails (actual emails themselves):

```
account/email/email_confirmation_message.txt
account/email/email_confirmation_subject.txt
account/email/invite_user.txt
account/email/invite_user_subject.txt
account/email/password_change.txt
account/email/password_change_subject.txt
account/email/password_reset.txt
account/email/password_reset_subject.txt
```

Template Tags

To use the built in template tags you must first load them within the templates:

```
{% load account_tags %}
```

To display the current logged-in user:

```
{% user_display request.user %}
```

Signals

user_signed_up

Triggered when a user signs up successfully. Providing arguments `user` (User instance) and `form` (form instance) as arguments.

user_sign_up_attempt

Triggered when a user tried but failed to sign up. Providing arguments `username` (string), `email` (string) and `result` (boolean, False if the form did not validate).

user_logged_in

Triggered when a user logs in successfully. Providing arguments `user` (User instance) and `form` (form instance).

user_login_attempt

Triggered when a user tries and fails to log in. Providing arguments `username` (string) and `result` (boolean, False if the form did not validate).

signup_code_sent

Triggered when a signup code was sent. Providing argument `signup_code` (SignupCode instance).

signup_code_used

Triggered when a user used a signup code. Providing argument `signup_code_result` (SignupCodeResult instance).

email_confirmed

Triggered when a user confirmed an email. Providing argument ‘‘email_address‘‘(EmailAddress instance).

email_confirmation_sent

Triggered when an email confirmation was sent. Providing argument `confirmation` (EmailConfirmation instance).

password_changed

Triggered when a user changes his password. Providing argument `user` (User instance).

password_expired

Triggered when a user password is expired. Providing argument `user` (User instance).

Management Commands

user_password_history

Creates an initial password history for all users who don't already have a password history.

Accepts two optional arguments:

```
-d --days <days> - Sets the age of the current password. Default is 10 days.
-f --force - Sets a new password history for ALL users, regardless of prior history.
```

user_password_expiry

Creates a password expiry specific to one user.

Password expiration checks use a global value (`ACCOUNT_PASSWORD_EXPIRY`) for the expiration time period. This value can be superseded on a per-user basis by creating a user password expiry.

Requires one argument:

```
<username> [<username>] - username(s) of the user(s) who needs specific password␣
↪expiry.
```

Accepts one optional argument:

```
-e --expire <seconds> - Sets the number of seconds for password expiration.
                        Default is the current global ACCOUNT_PASSWORD_EXPIRY value.
```

After creation, you can modify user password expiration from the Django admin. Find the desired user at `/admin/account/passwordexpiry/` and change the `expiry` value.

Migration from Pinax

django-user-accounts is based on `pinax.apps.account` combining some of the supporting apps. django-email-confirmation, `pinax.apps.signup_codes` and bits of django-timezones have been merged to create django-user-accounts.

This document will outline the changes needed to migrate from Pinax to using this app in your Django project. If you are new to django-user-accounts then this guide will not be useful to you.

Database changes

Due to combining apps the table layout when converting from Pinax has changed. We've also taken the opportunity to update the schema to take advantage of much saner defaults. Here is SQL to convert from Pinax to django-user-accounts.

PostgreSQL

```
ALTER TABLE "signup_codes_signupcode" RENAME TO "account_signupcode";
ALTER TABLE "signup_codes_signupcoderesult" RENAME TO "account_signupcoderesult";
ALTER TABLE "emailconfirmation_emailaddress" RENAME TO "account_emailaddress";
ALTER TABLE "emailconfirmation_emailconfirmation" RENAME TO "account_emailconfirmation
↪";
DROP TABLE "account_passwordreset";
ALTER TABLE "account_signupcode" ALTER COLUMN "code" TYPE varchar(64);
ALTER TABLE "account_signupcode" ADD CONSTRAINT "account_signupcode_code_key" UNIQUE (
↪"code");
ALTER TABLE "account_emailconfirmation" RENAME COLUMN "confirmation_key" TO "key";
ALTER TABLE "account_emailconfirmation" ALTER COLUMN "key" TYPE varchar(64);
ALTER TABLE account_emailconfirmation ADD COLUMN created timestamp with time zone;
UPDATE account_emailconfirmation SET created = sent;
ALTER TABLE account_emailconfirmation ALTER COLUMN created SET NOT NULL;
ALTER TABLE account_emailconfirmation ALTER COLUMN sent DROP NOT NULL;
```

If `ACCOUNT_EMAIL_UNIQUE` is set to `True` (the default value) you need:

```
ALTER TABLE "account_emailaddress" ADD CONSTRAINT "account_emailaddress_email_key"␣
↪UNIQUE ("email");
ALTER TABLE "account_emailaddress" DROP CONSTRAINT "emailconfirmation_emailaddress_
↪user_id_email_key";
```

MySQL

```
RENAME TABLE `emailconfirmation_emailaddress` TO `account_emailaddress` ;
RENAME TABLE `emailconfirmation_emailconfirmation` TO `account_emailconfirmation` ;
DROP TABLE account_passwordreset;
ALTER TABLE `account_emailconfirmation` CHANGE `confirmation_key` `key`␣
↪VARCHAR(64) NOT NULL;
ALTER TABLE `account_emailconfirmation` ADD UNIQUE (`key`);
ALTER TABLE account_emailconfirmation ADD COLUMN created datetime NOT NULL;
UPDATE account_emailconfirmation SET created = sent;
ALTER TABLE `account_emailconfirmation` CHANGE `sent` `sent` DATETIME NULL;
```

If `ACCOUNT_EMAIL_UNIQUE` is set to `True` (the default value) you need:

```
ALTER TABLE `account_emailaddress` ADD UNIQUE (`email`);
ALTER TABLE account_emailaddress DROP INDEX user_id;
```

If you have installed `pinax.apps.signup_codes`:

```
RENAME TABLE `signup_codes_signupcode` TO `account_signupcode` ;
RENAME TABLE `signup_codes_signupcoderesult` TO `account_signupcoderesult` ;
```

URL changes

Here is a list of all URLs provided by django-user-accounts and how they map from Pinax. This assumes `account.urls` is mounted at `/account/` as it was in Pinax.

Pinax	django-user-accounts
`/account/login/`	`/account/login/`
`/account/signup/`	`/account/signup/`
`/account/confirm_email/`	`/account/confirm_email/`
`/account/password_change/`	`/account/password/`[1]
`/account/password_reset/`	`/account/password/reset/`
`/account/password_reset_done/`	*removed*
`/account/password_reset_key/<key>/`	`/account/password/reset/<token>/`

View changes

All views have been converted to class-based views. This is a big departure from the traditional function-based, but has the benefit of being much more flexible.

@@@ todo: table of changes

Settings changes

We have cleaned up settings and set saner defaults used by django-user-accounts.

Pinax	django-user-accounts
`ACCOUNT_OPEN_SIGNUP = True`	`ACCOUNT_OPEN_SIGNUP = True`
`ACCOUNT_UNIQUE_EMAIL = False`	`ACCOUNT_EMAIL_UNIQUE = True`
`EMAIL_CONFIRMATION_UNIQUE_EMAIL = False`	*removed*

General changes

django-user-accounts requires Django 1.4. This means we can take advantage of many of the new features offered by Django. This app implements all of the best practices of Django 1.4. If there is something missing you should let us know!

FAQ

This document is a collection of frequently asked questions about django-user-accounts.

What is the difference between django-user-accounts and django.contrib.auth?

django-user-accounts is designed to supplement `django.contrib.auth`. This app provides improved views for log in, password reset, log out and adds sign up functionality. We try not to duplicate code when Django provides a good implementation. For example, we did not re-implement password reset, but simply provide an improved view which calls into the secure Django password reset code. `django.contrib.auth` is still providing many of supporting elements such as `User` model, default authentication backends, helper functions and authorization.

django-user-accounts takes your Django project from having simple log in, log out and password reset to a full blown account management system that you will end up building anyways.

[1] When user is anonymous and requests a GET the user is redirected to `/account/password/reset/`.

Why can email addresses get out of sync?

django-user-accounts stores email addresses in two locations. The default `User` model contains an `email` field and django-user-accounts provides an `EmailAddress` model. This latter is provided to support multiple email addresses per user.

If you use a custom user model you can prevent the double storage. This is because you can choose not to do any email address storage.

If you don't use a custom user model then make sure you take extra precaution. When editing email addresses either in the shell or admin make sure you update in both places. Only the primary email address is stored on the `User` model.