django-openid Documentation

Release 2.0a

Simon Willison

September 27, 2017

Contents

django_openid provides everything you need to handle OpenID logins in your Django applications. The default settings are designed to get you up and running as quickly as possible, while finely grained extension hooks allow advanced users to customise every aspect of the OpenID flow.

CHAPTER 1

Installation

1. Install python-openid version 2.2.1 or later.
2. Check out django_openid somewhere on your Python path:

```
svn co http://django-openid.googlecode.com/svn/trunk/django_openid django_openid
```

3. Add `'django_openid'` to the `INSTALLED_APPS` setting for your project.
4. Run `./manage.py syncdb` to install the database tables required by django_openid. (It is possible to run django_openid without using a database at all; see elsewhere in the documentation for details).

CHAPTER 2

Accepting OpenID

If you just want users to be able to sign in to your application with an OpenID, you have two options: SessionConsumer and CookieConsumer. SessionConsumer uses Django's session framework; most people will probably want to use this. If you don't want to use Django's sessions, CookieConsumer is an alternative that uses signed cookies instead.

Add the following to your urls.py:

```
from django_openid.consumer import SessionConsumer

urlpatterns = patterns('',
    # ...
    (r'^openid/(.*)', SessionConsumer()),
)
```

You'll need to have Django's session framework installed.

Now, if you visit yoursite.example.com/openid/ you will be presented with an OpenID login form. Sign in with OpenID and you will be redirected back to your site's homepage. An OpenID object representing your OpenID will be stored in the session, in request.session['openids'][0]

If you sign in again with a different OpenID, it will be appended to the end of the request.session['openids'] list.

You can access the authenticated OpenID as a unicode string with the following:

```
request.session['openids'][-1].openid
```

For a more convenient API for accessing the OpenID, enable the SessionConsumer Django middleware. Add the following to your MIDDLEWARE_CLASSES setting:

```
MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    # ...
    'django_openid.consumer.SessionConsumer',
)
```

The SessionConsumer middleware must come after the SessionMiddleware as it needs access to the session.

With the middleware installed, you can access the user's OpenID using a property on the request object itself:

```
if request.openid:
    # The user is signed in with an OpenID
    return HttpResponse("You are signed in as %s" % request.openid.openid)
```

request.openids is a list of signed in OpenIDs, if the user has signed in with more than one.

To log out (deleting the OpenIDs from the session), simply visit /openid/logout/ on your site.

If you only want to sign out with one, specific OpenID use the following link instead:

/openid/logout/?openid=http://specific-openid.example.com/

Redirecting somewhere else

If you don't want to redirect your users to the homepage once they have logged in, you can customise the redirect URL with your own SessionConsumer subclass:

```
from django_openid.consumer import SessionConsumer

class MyConsumer(SessionConsumer):
    redirect_after_login = '/welcome/'
```

Then simply map that class in your urlconf instead:

```
urlpatterns = patterns('',
    # ...
    (r'^openid/(.*)', MyConsumer()),
)
```

Requesting sreg information about your user

The OpenID simple registration extension allows you to request additional profile information from your user's OpenID provider. Available fields are nickname, email, fullname, dob, gender, postcode, country, language, timezone.

Remember, while you can request any or all of those fields there are absolutely no guarantees that the OpenID provider will fulfill your request. This is despite the inclusion of so-called 'required' fields in the sreg specification. The best way to use sreg data is to pre-populate a sign-up form that is displayed to users with OpenIDs that you haven't seen before.

That said, requesting sreg information is easy - simply add an sreg property to your SessionConsumer subclass:

```
class MyConsumer(SessionConsumer):
    redirect_after_login = '/welcome/'
    sreg = ['nickname', 'email', 'dob', 'postcode']
```

Assuming your user's OpenID provider supports sreg and your user opts in to sending you that data, you'll be able to access it using the request.openid.sreg dictionary:

```
def view(request):
    if request.openid:
        nickname = request.openid.sreg.get('nickname', '')
```

```
        email = request.openid.sreg.get('email', '')
        # etc
```

Customising the templates

django_openid ships with default templates that are functional but plain looking. There are a number of different ways in which you can customise them.

All of the templates extend a common base template. By default, this is 'django_openid/base.html' which can be found in the django_openid/templates directory. You can over-ride this template by adding one with the same path to one of the directories in your TEMPLATE_DIRS setting. You'll need to define blocks called "title" and "content", fitting the Django convention.

If you already have a base template for your project that fits these naming conventions, you can use it by setting the base_template attribute on your custom subclass:

```
class MyConsumer(SessionConsumer):
    redirect_after_login = '/welcome/'
    sreg = ['nickname', 'email', 'dob', 'postcode']

    base_template = 'path/to/base.html'
```

django_openid provides plenty of other class attributes that you can over-ride in your subclass, including attributes for selecting different templates for different views and attributes for customising the default error messages shown to your users. Explore the source code for details.

Using regular cookies instead of sessions

If you don't want to use Django's built-in session support you can still use django_openid in much the same way as with the SessionConsumer, thanks to the CookieConsumer class. It is configured in exactly the same way:

```
from django_openid.consumer import CookieConsumer

class MyConsumer(CookieConsumer):
    redirect_after_login = '/welcome/'
    sreg = ['nickname', 'email', 'dob', 'postcode']

    base_template = 'path/to/base.html'

# Then in your settings.py

MIDDLEWARE_CLASSES = (
    # ...
    'django_openid.consumer.CookieConsumer',
)
```

The CookieConsumer uses signed cookies to store the user's OpenID, including their sreg information. Unlike the SessionConsumer, the CookieConsumer only allows users to be signed in with one OpenID at a time. The middleware provides request.openid and request.openids properties that are identical to SessionConsumer, but request.openids will only ever contain 0 or 1 OpenID objects.

Using django_openid without a database

Under the hood, django_openid uses the JanRain OpenID library. This library needs to store book-keeping information about the current active OpenID request somewhere. By default, django_openid stores this in the database - that's why you need to run ./manage.py syncdb when you install it.

CHAPTER 3

Integrating with django.contrib.auth

The obvious next step with OpenID is to integrate it with Django's built-in concept of authentication, using the models from django.contrib.auth (in particular the User) model. The correct way of thinking about OpenID in this context is as an alternative to authenticating with a password. django_openid supports allowing users to associate 0 or more OpenIDs with a User account.

Setting up auth integration

Auth integration is implemented using AuthConsumer, a subclass of Consumer. AuthConsumer adds the ability to associate OpenIDs with user accounts.

If you want users to be able to register for new accounts on your site using their OpenID, you should use RegistrationConsumer instead. RegistrationConsumer subclasses AuthConsumer but adds a flow for registering new accounts.

Here's how to set up AuthConsumer:

```
from django.conf.urls.defaults import *
from django_openid.registration import RegistrationConsumer

urlpatterns = patterns('',
    # ...
    (r'^openid/(.*)', RegistrationConsumer()),
    # ...
)
```

If you are using Django 1.1, you can do the following instead:

```
from django.conf.urls.defaults import *
from django_openid.registration import RegistrationConsumer

registration_consumer = Consumer()

urlpatterns = patterns('',
```

```
    (r'^openid/', include(registration_consumer.urls)),
)
```

Using named URL patterns

Using Django 1.1 and the include pattern shown above, URLs within the registration consumer will be exposed as named URL patterns. By default, the names will follow the pattern 'openid-ACTION' - but you can change this default if you like by over-riding the urlname_pattern property of your Consumer subclass.

You can also provide names to specific patterns using the following idiom (which also works in Django 1.0):

```
url(r'^account/register/$', registration_consumer, {
    'rest_of_url': 'register/',
}, name = 'custom-register'),
url(r'^account/login/$', registration_consumer, {
    'rest_of_url': 'login/',
}, name = 'custom-login'),
url(r'^account/logout/$', registration_consumer, {
    'rest_of_url': 'logout/',
}, name = 'custom-logout'),
(r'^account/(.*?)$', registration_consumer),
```

You can also use this idiom to apply decorators to individual paths within the Consumer:

```
url(r'^account/register/$', view_decorator(registration_consumer), {
    'rest_of_url': 'register/',
}, name = 'custom-register'),
```

CHAPTER 4

API

django_openid.consumer

Accepting OpenID logins.

django_openid.auth

Integrating OpenID with Django auth.

django_openid.registration

Allow users to register new user accounts with or without an OpenID.

django_openid.provider

Implement an OpenID provider.

CHAPTER 5

Understanding the registration flow

The full flow for a site that offers login and registration using either OpenID or a regular username/password account can be quite complicated. This page shows how the flow implemented by django-openid works.

Login

- **With Username and Password:**
  - (has link to switch to OpenID)
  - **Incorrect login:**
    - **They get their username wrong:**
      - "wrong username" message (defaults to same as wrong password)
    - **They get their password wrong:**
      - "wrong password" message
  - **Correct login:**
    - **Have they verified their e-mail address?**
      - **Yes:**

        Log them in to that account.
      - **No:**

        Tell them to verify their e-mail address

        Option: send me that e-mail again
- **With OpenID:**
  - (has link to switch to username/password)
  - **OpenID is invalid or authentication fails:**

        * Tell them what happened, don't log them in
        * Show login or register UI
    - **OpenID is valid and corresponds to an existing account:**
        * **Have they verified their e-mail address?**
            - **Yes:**

              Log them in to that account.
            - **No:**

              Tell them to verify their e-mail address

              Option: send me that e-mail again
    - **OpenID is valid but does not correspond to an existing account:**
        * Tell them, and offer a link to the registration form.

Registration

- **Register with username/password:**
    - Username/e-mail/password please
    - Repeat until valid
    - Send verification e-mail
    - Tell them "just one more step: click link in your e-mail"
- **Register with OpenID:**
    - **Enter your OpenID:**
        * **OpenID is valid:**
            - **And not associated with existing account:**

              Show registration form, pre-filled with any details from OpenID provider
            - **And associated with existing account:**

              Log them in - jump to "have they verified their e-mail address" bit in login with OpenID flow
        * **OpenID is invalid:**
            - Tell them what happened, link to login page

E-mail verification link clicked

- **Is verification code valid?**
    - **Yes:**
        * Mark that account as verified
        * Log them straight in to that account
    - **No:**

        * Tell them it's invalid.
        * Provide a link to the homepage.

Recover account flow

- **Ask them for their:**
    - E-mail address
    - or Username
    - or OpenID
- **If valid information:**
    - Send them an e-mail with a magic log-you-in link in it
- **If invalid:**
    - Tell them no account found.
    - Show form again.

Recover link clicked

- **If valid:**
    - Log them in
    - Optionally show a reset-your-password screen
- **If invalid:**
    - Tell them to go away, link to homepage

CHAPTER 6

Design notes

The thoughts behind various aspects of the design of django-openid.

Confirmation e-mails

- Not all sites wish to implement a confirm-via-email loop (which can discourage people from signing up) so it should not be a compulsory feature.
- People sometimes lose confirmation e-mails to spam filters and so forth - they need to be able to request that an e-mail is re-sent.
- **It's important to be able to distinguish between users who have not yet confirmed their account and users who have been l**
  - Solution: the "Unconfirmed users" group is used to mark accounts which have not yet been confirmed. Only accounts in that group are allowed to re-request confirmation e-mails.

CHAPTER 7

Features

Using django_openid, you can:

- Ask a user to **log in with OpenID**, and perform a custom action once their identity has been confirmed.
- **Persist a user's identity** (using either sessions or a signed cookie) so that they can log in once with their OpenID before accessing other pages of your site.
- **Request additional information** about the user from their OpenID provider, such as their preferred nickname, e-mail address or date of birth, using the sreg OpenID extension.
- Accept **both versions 1.1 and 2.0** of the OpenID protocol.
- Allow users to **sign in to their django.contrib.auth User accounts** using an OpenID, or **associate one or more OpenIDs** with their existing account.
- Give your users OpenIDs hosted by your own site, by acting as an **OpenID provider**.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

D