django-allauth Documentation

Release 0.32.0

Raymond Penners

Sep 23, 2017

Contents

Integrated set of Django applications addressing authentication, registration, account management as well as 3rd party (social) account authentication.

Home page http://www.intenct.nl/projects/django-allauth/

Source code http://github.com/pennersr/django-allauth

Mailinglist http://groups.google.com/group/django-allauth

Documentation https://django-allauth.readthedocs.io/en/latest/

Stack Overflow http://stackoverflow.com/questions/tagged/django-allauth

CHAPTER 1

Rationale

Most existing Django apps that address the problem of social authentication focus on just that. You typically need to integrate another app in order to support authentication via a local account.

This approach separates the worlds of local and social authentication. However, there are common scenarios to be dealt with in both worlds. For example, an e-mail address passed along by an OpenID provider is not guaranteed to be verified. So, before hooking an OpenID account up to a local account the e-mail address must be verified. So, e-mail verification needs to be present in both worlds.

Integrating both worlds is quite a tedious process. It is definitely not a matter of simply adding one social authentication app, and one local account registration app to your `INSTALLED_APPS` list.

This is the reason this project got started – to offer a fully integrated authentication app that allows for both local and social authentication, with flows that just work.

CHAPTER 2

Commercial Support

This project is sponsored by IntenCT. If you require assistance on your project(s), please contact us: info@intenct.nl.

CHAPTER 3

Cross-Selling

If you like this, you may also like:

- django-trackstats: https://github.com/pennersr/django-trackstats
- netwell: https://github.com/pennersr/netwell

CHAPTER 4

Contents

Overview

Requirements

- Python 2.7, 3.3, 3.4, or 3.5
- Django (1.8+)
- python-openid or python3-openid (depending on your Python version)
- requests and requests-oauthlib

Supported Flows

- Signup of both local and social accounts
- Connecting more than one social account to a local account
- Disconnecting a social account – requires setting a password if only the local account remains
- Optional instant-signup for social accounts – no questions asked
- E-mail address management (multiple e-mail addresses, setting a primary)
- Password forgotten flow
- E-mail address verification flow

Supported Providers

- 23andMe (OAuth2)
- 500px
- Amazon (OAuth2)

- AngelList (OAuth2)
- Asana (OAuth2)
- Auth0 (OAuth2)
- Authentiq (OAuth2)
- Basecamp (OAuth2)
- Baidu (OAuth2)
- Battle.net (OAuth2)
- Bitbucket (OAuth, OAuth2)
- Bitly (OAuth2)
- Box (OAuth2)
- Dataporten (OAuth2)
- Daum (OAuth2)
- Douban (OAuth2)
- Doximity (OAuth2)
- Dropbox (OAuth, OAuth2)
- Dwolla (OAuth2)
- Edmodo (OAuth2)
- Eve Online (OAuth2)
- Eventbrite (OAuth2)
- Evernote (OAuth)
- Facebook (both OAuth2 and JS SDK)
- Feedly (OAuth2)
- Firefox Accounts (OAuth2)
- Flickr (OAuth)
- Github (OAuth2)
- GitLab (OAuth2)
- Google (OAuth2)
- Hubic (OAuth2)
- Instagram (OAuth2)
- Kakao (OAuth2)
- Line (OAuth2)
- LinkedIn (OAuth, OAuth2)
- Mail.Ru (OAuth2)
- MailChimp (OAuth2)
- Naver (OAuth2)
- Odnoklassniki (OAuth2)

- OpenId
- ORCID (OAuth2)
- Patreon (OAuth2)
- Paypal (OAuth2)
- Persona
- Pinterest (OAuth2)
- Reddit (OAuth2)
- Shopify (OAuth2)
- Slack (OAuth2)
- SoundCloud (OAuth2)
- Spotify (OAuth2)
- Stack Exchange (OAuth2)
- Stripe (OAuth2)
- Trello (OAuth)
- Tumblr (OAuth)
- Twitch (OAuth2)
- Twitter (OAuth)
- Untappd (OAuth2)
- Vimeo (OAuth)
- VK (OAuth2)
- Weibo (OAuth2)
- Weixin (OAuth2)
- Windows Live (OAuth2)
- Xing (OAuth)

Note: OAuth/OAuth2 support is built using a common code base, making it easy to add support for additional OAuth/OAuth2 providers. More will follow soon...

Features

- Supports multiple authentication schemes (e.g. login by user name, or by e-mail), as well as multiple strategies for account verification (ranging from none to e-mail verification).
- All access tokens are consistently stored so that you can publish wall updates etc.

Architecture & Design

- Pluggable signup form for asking additional questions during signup.
- Support for connecting multiple social accounts to a Django user account.
- The required consumer keys and secrets for interacting with Facebook, Twitter and the likes are to be configured in the database via the Django admin using the SocialApp model.

- Consumer keys, tokens make use of the Django sites framework. This is especially helpful for larger multi-domain projects, but also allows for for easy switching between a development (localhost) and production setup without messing with your settings and database.

Installation

Django

Python package:

```
pip install django-allauth
```

settings.py (Important - Please note 'django.contrib.sites' is required as INSTALLED_APPS):

```
# Specify the context processors as follows:
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                # Already defined Django-related contexts here

                # `allauth` needs this from django
                'django.template.context_processors.request',
            ],
        },
    },
]

AUTHENTICATION_BACKENDS = (
    ...
    # Needed to login by username in Django admin, regardless of `allauth`
    'django.contrib.auth.backends.ModelBackend',

    # `allauth` specific authentication methods, such as login by e-mail
    'allauth.account.auth_backends.AuthenticationBackend',
    ...
)

INSTALLED_APPS = (
    ...
    # The following apps are required:
    'django.contrib.auth',
    'django.contrib.sites',

    'allauth',
    'allauth.account',
    'allauth.socialaccount',
    # ... include the providers you want to enable:
    'allauth.socialaccount.providers.amazon',
    'allauth.socialaccount.providers.angellist',
    'allauth.socialaccount.providers.asana',
    'allauth.socialaccount.providers.auth0',
    'allauth.socialaccount.providers.authentiq',
```

```
    'allauth.socialaccount.providers.baidu',
    'allauth.socialaccount.providers.basecamp',
    'allauth.socialaccount.providers.bitbucket',
    'allauth.socialaccount.providers.bitbucket_oauth2',
    'allauth.socialaccount.providers.bitly',
    'allauth.socialaccount.providers.coinbase',
    'allauth.socialaccount.providers.dataporten',
    'allauth.socialaccount.providers.daum',
    'allauth.socialaccount.providers.digitalocean',
    'allauth.socialaccount.providers.discord',
    'allauth.socialaccount.providers.douban',
    'allauth.socialaccount.providers.draugiem',
    'allauth.socialaccount.providers.dropbox',
    'allauth.socialaccount.providers.dropbox_oauth2',
    'allauth.socialaccount.providers.dwolla',
    'allauth.socialaccount.providers.edmodo',
    'allauth.socialaccount.providers.eveonline',
    'allauth.socialaccount.providers.evernote',
    'allauth.socialaccount.providers.facebook',
    'allauth.socialaccount.providers.feedly',
    'allauth.socialaccount.providers.fivehundredpx',
    'allauth.socialaccount.providers.flickr',
    'allauth.socialaccount.providers.foursquare',
    'allauth.socialaccount.providers.fxa',
    'allauth.socialaccount.providers.github',
    'allauth.socialaccount.providers.gitlab',
    'allauth.socialaccount.providers.google',
    'allauth.socialaccount.providers.hubic',
    'allauth.socialaccount.providers.instagram',
    'allauth.socialaccount.providers.kakao',
    'allauth.socialaccount.providers.line',
    'allauth.socialaccount.providers.linkedin',
    'allauth.socialaccount.providers.linkedin_oauth2',
    'allauth.socialaccount.providers.mailru',
    'allauth.socialaccount.providers.mailchimp',
    'allauth.socialaccount.providers.naver',
    'allauth.socialaccount.providers.odnoklassniki',
    'allauth.socialaccount.providers.openid',
    'allauth.socialaccount.providers.orcid',
    'allauth.socialaccount.providers.paypal',
    'allauth.socialaccount.providers.persona',
    'allauth.socialaccount.providers.pinterest',
    'allauth.socialaccount.providers.reddit',
    'allauth.socialaccount.providers.robinhood',
    'allauth.socialaccount.providers.shopify',
    'allauth.socialaccount.providers.slack',
    'allauth.socialaccount.providers.soundcloud',
    'allauth.socialaccount.providers.spotify',
    'allauth.socialaccount.providers.stackexchange',
    'allauth.socialaccount.providers.stripe',
    'allauth.socialaccount.providers.trello',
    'allauth.socialaccount.providers.tumblr',
    'allauth.socialaccount.providers.twentythreeandme',
    'allauth.socialaccount.providers.twitch',
    'allauth.socialaccount.providers.twitter',
    'allauth.socialaccount.providers.untappd',
    'allauth.socialaccount.providers.vimeo',
    'allauth.socialaccount.providers.vk',
```

```
    'allauth.socialaccount.providers.weibo',
    'allauth.socialaccount.providers.weixin',
    'allauth.socialaccount.providers.windowslive',
    'allauth.socialaccount.providers.xing',
    ...
)

SITE_ID = 1
```

urls.py:

```
urlpatterns = [
    ...
    url(r'^accounts/', include('allauth.urls')),
    ...
]
```

Note that you do not necessarily need the URLs provided by `django.contrib.auth.urls`. Instead of the URLs `login`, `logout`, and `password_change` (among others), you can use the URLs provided by `allauth`: `account_login`, `account_logout`, `account_set_password`...

Post-Installation

In your Django root execute the command below to create your database tables:

```
./manage.py migrate
```

Now start your server, visit your admin pages (e.g. http://localhost:8000/admin/) and follow these steps:

1. Add a `Site` for your domain, matching `settings.SITE_ID` (`django.contrib.sites` app).
2. For each OAuth based provider, add a `Social App` (`socialaccount` app).
3. Fill in the site and the OAuth app credentials obtained from the provider.

Configuration

Available settings:

ACCOUNT_ADAPTER (="allauth.account.adapter.DefaultAccountAdapter") Specifies the adapter class to use, allowing you to alter certain default behaviour.

ACCOUNT_AUTHENTICATED_LOGIN_REDIRECTS (=True) The default behaviour is to redirect authenticated users to `LOGIN_REDIRECT_URL` when they try accessing login/signup pages.

By changing this setting to `False`, logged in users will not be redirected when they access login/signup pages.

ACCOUNT_AUTHENTICATION_METHOD (="username" | "email" | "username_email") Specifies the login method to use – whether the user logs in by entering their username, e-mail address, or either one of both. Setting this to "email" requires ACCOUNT_EMAIL_REQUIRED=True

ACCOUNT_CONFIRM_EMAIL_ON_GET (=False) Determines whether or not an e-mail address is automatically confirmed by a GET request. GET is not designed to modify the server state, though it is commonly used for email confirmation. To avoid requiring user interaction, consider using POST via Javascript in your email confirmation template as an alternative to setting this to True.

ACCOUNT_EMAIL_CONFIRMATION_ANONYMOUS_REDIRECT_URL (=settings.LOGIN_URL) The URL to redirect to after a successful e-mail confirmation, in case no user is logged in.

ACCOUNT_EMAIL_CONFIRMATION_AUTHENTICATED_REDIRECT_URL (=None) The URL to redirect to after a successful e-mail confirmation, in case of an authenticated user. Set to `None` to use `settings.LOGIN_REDIRECT_URL`.

ACCOUNT_EMAIL_CONFIRMATION_EXPIRE_DAYS (=3) Determines the expiration date of email confirmation mails (# of days).

ACCOUNT_EMAIL_CONFIRMATION_HMAC (=True) In order to verify an email address a key is mailed identifying the email address to be verified. In previous versions, a record was stored in the database for each ongoing email confirmation, keeping track of these keys. Current versions use HMAC based keys that do not require server side state.

ACCOUNT_EMAIL_REQUIRED (=False) The user is required to hand over an e-mail address when signing up.

ACCOUNT_EMAIL_VERIFICATION (="optional") Determines the e-mail verification method during signup – choose one of `"mandatory"`, `"optional"`, or `"none"`. When set to "mandatory" the user is blocked from logging in until the email address is verified. Choose "optional" or "none" to allow logins with an unverified e-mail address. In case of "optional", the e-mail verification mail is still sent, whereas in case of "none" no e-mail verification mails are sent.

ACCOUNT_EMAIL_SUBJECT_PREFIX (="[Site] ") Subject-line prefix to use for email messages sent. By default, the name of the current `Site` (`django.contrib.sites`) is used.

ACCOUNT_DEFAULT_HTTP_PROTOCOL (="http") The default protocol used for when generating URLs, e.g. for the password forgotten procedure. Note that this is a default only – see the section on HTTPS for more information.

ACCOUNT_EMAIL_CONFIRMATION_COOLDOWN (=180) The cooldown period (in seconds) after a confirmation email is sent, during which further emails are not sent.

ACCOUNT_FORMS (={}) Used to override forms, for example: `{'login': 'myapp.forms.LoginForm'}`

ACCOUNT_LOGIN_ATTEMPTS_LIMIT (=5) Number of failed login attempts. When this number is exceeded, the user is prohibited from logging in for the specified `ACCOUNT_LOGIN_ATTEMPTS_TIMEOUT` seconds. Set to `None` to disable this functionality. Important: while this protects the allauth login view, it does not protect Django's admin login from being brute forced.

ACCOUNT_LOGIN_ATTEMPTS_TIMEOUT (=300) Time period, in seconds, from last unsuccessful login attempt, during which the user is prohibited from trying to log in.

ACCOUNT_LOGIN_ON_EMAIL_CONFIRMATION (=False) The default behaviour is not log users in and to redirect them to `ACCOUNT_EMAIL_CONFIRMATION_ANONYMOUS_REDIRECT_URL`.

By changing this setting to `True`, users will automatically be logged in once they confirm their email address. Note however that this only works when confirming the email address **immediately after signing up**, assuming users didn't close their browser or used some sort of private browsing mode.

ACCOUNT_LOGOUT_ON_GET (=False) Determines whether or not the user is automatically logged out by a GET request. GET is not designed to modify the server state, and in this case it can be dangerous. See LogoutView in the documentation for details.

ACCOUNT_LOGOUT_ON_PASSWORD_CHANGE (=False) Determines whether or not the user is automatically logged out after changing or setting their password. See documentation for Django's session invalidation on password change.

ACCOUNT_LOGIN_ON_PASSWORD_RESET (=False) By changing this setting to `True`, users will automatically be logged in once they have reset their password. By default they are redirected to the password reset done page.

ACCOUNT_LOGOUT_REDIRECT_URL (="/") The URL (or URL name) to return to after the user logs out. This is the counterpart to Django's `LOGIN_REDIRECT_URL`.

ACCOUNT_PASSWORD_INPUT_RENDER_VALUE (=False) `render_value` parameter as passed to `PasswordInput` fields.

ACCOUNT_PRESERVE_USERNAME_CASING (=True) This setting determines whether the username is stored in lowercase (`False`) or whether its casing is to be preserved (`True`). Note that when casing is preserved, potentially expensive `__iexact` lookups are performed when filter on username. For now, the default is set to `True` to maintain backwards compatibility.

ACCOUNT_SESSION_REMEMBER (=None) Controls the life time of the session. Set to `None` to ask the user ("Remember me?"), `False` to not remember, and `True` to always remember.

ACCOUNT_SIGNUP_EMAIL_ENTER_TWICE (=False) When signing up, let the user type in their email address twice to avoid typo's.

ACCOUNT_SIGNUP_FORM_CLASS (=None) A string pointing to a custom form class (e.g. 'myapp.forms.SignupForm') that is used during signup to ask the user for additional input (e.g. newsletter signup, birth date). This class should implement a `def signup(self, request, user)` method, where user represents the newly signed up user.

ACCOUNT_SIGNUP_PASSWORD_ENTER_TWICE (=True) When signing up, let the user type in their password twice to avoid typos.

ACCOUNT_TEMPLATE_EXTENSION (="html") A string defining the template extension to use, defaults to `html`.

ACCOUNT_USERNAME_BLACKLIST (=[]) A list of usernames that can't be used by user.

ACCOUNT_UNIQUE_EMAIL (=True) Enforce uniqueness of e-mail addresses. The `emailaddress.email` model field is set to `UNIQUE`. Forms prevent a user from registering with or adding an additional email address if that email address is in use by another account.

ACCOUNT_USER_DISPLAY (=a callable returning `user.username`) A callable (or string of the form `'some.module.callable_name'`) that takes a user as its only argument and returns the display name of the user. The default implementation returns `user.username`.

ACCOUNT_USER_MODEL_EMAIL_FIELD (="email") The name of the field containing the `email`, if any. See custom user models.

ACCOUNT_USER_MODEL_USERNAME_FIELD (="username") The name of the field containing the `username`, if any. See custom user models.

ACCOUNT_USERNAME_MIN_LENGTH (=1) An integer specifying the minimum allowed length of a username.

ACCOUNT_USERNAME_REQUIRED (=True) The user is required to enter a username when signing up. Note that the user will be asked to do so even if `ACCOUNT_AUTHENTICATION_METHOD` is set to `email`. Set to `False` when you do not wish to prompt the user to enter a username.

ACCOUNT_USERNAME_VALIDATORS (=None) A path (`'some.module.validators.custom_username_validators'`) to a list of custom username validators. If left unset, the validators setup within the user model username field are used.

Example:

```
# In validators.py

from django.contrib.auth.validators import ASCIIUsernameValidator

custom_username_validators = [ASCIIUsernameValidator()]
```

```
# In settings.py

ACCOUNT_USERNAME_VALIDATORS = 'some.module.validators.custom_username_validators'
```

SOCIALACCOUNT_ADAPTER (="allauth.socialaccount.adapter.DefaultSocialAccountAdapter") Specifies the adapter class to use, allowing you to alter certain default behaviour.

SOCIALACCOUNT_AUTO_SIGNUP (=True) Attempt to bypass the signup form by using fields (e.g. username, email) retrieved from the social account provider. If a conflict arises due to a duplicate e-mail address the signup form will still kick in.

SOCIALACCOUNT_EMAIL_VERIFICATION (=ACCOUNT_EMAIL_VERIFICATION) As `ACCOUNT_EMAIL_VERIFICATION`, but for social accounts.

SOCIALACCOUNT_EMAIL_REQUIRED (=ACCOUNT_EMAIL_REQUIRED) The user is required to hand over an e-mail address when signing up using a social account.

SOCIALACCOUNT_FORMS (={}) Used to override forms, for example: `{'signup': 'myapp.forms.SignupForm'}`

SOCIALACCOUNT_PROVIDERS (= dict) Dictionary containing provider specific settings.

SOCIALACCOUNT_QUERY_EMAIL (=ACCOUNT_EMAIL_REQUIRED) Request e-mail address from 3rd party account provider? E.g. using OpenID AX, or the Facebook "email" permission.

SOCIALACCOUNT_STORE_TOKENS (=True) Indicates whether or not the access tokens are stored in the database.

Providers

Most providers require you to sign up for a so called API client or app, containing a client ID and API secret. You must add a `SocialApp` record per provider via the Django admin containing these app credentials.

When creating the OAuth app on the side of the provider pay special attention to the callback URL (sometimes also referred to as redirect URL). If you do not configure this correctly, you will receive login failures when attempting to log in, such as:

```
An error occurred while attempting to login via your social network account.
```

Use a callback URL of the form:

```
http://example.com/accounts/twitter/login/callback/
http://example.com/accounts/soundcloud/login/callback/
...
```

For local development, use the following:

```
http://127.0.0.1:8000/accounts/twitter/login/callback/
```

23andMe

App registration (get your key and secret here) https://api.23andme.com/dev/

Development callback URL http://localhost:8000/accounts/23andme/login/callback/

500px

App registration (get your key and secret here) https://500px.com/settings/applications

Development callback URL http://localhost:8000/accounts/500px/login/callback/

Amazon

Amazon requires secure OAuth callback URLs (`redirect_uri`), please see the section on HTTPS about how this is handled.

App registration (get your key and secret here) http://login.amazon.com/manageApps

Development callback URL https://example.com/accounts/amazon/login/callback/

AngelList

App registration (get your key and secret here) https://angel.co/api/oauth/clients

Development callback URL http://localhost:8000/accounts/angellist/login/callback/

Auth0

App registration (get your key and secret here) https://manage.auth0.com/#/clients

Development callback URL http://localhost:8000/accounts/auth0/login/callback/

You'll need to specify the base URL for your Auth0 domain:

```
SOCIALACCOUNT_PROVIDERS = {
    'auth0': {
        'AUTH0_URL': 'https://your.auth0domain.auth0.com',
    }
}
```

Authentiq

Browse to https://www.authentiq.com/developers to get started.

App registration https://dashboard.authentiq.com/

Sign in or register with your Authentiq ID (select `Download the app` while signing in if you don't have Authentiq ID yet).

Development redirect URL http://localhost:8000/accounts/authentiq/login/callback/

While testing you can leave the `Redirect URIs` field empty in the dashboard. You can specify what identity details to request via the `SCOPE` parameter.

```
SOCIALACCOUNT_PROVIDERS = {
    'authentiq': {
      'SCOPE': ['email', 'aq:name']
    }
}
```

Valid scopes include: `email`, `phone`, `address`, `aq:name`, `aq:location`. The default is to request a user's name, and email address if `SOCIALACCOUNT_QUERY_EMAIL=True`. You can request and require a verified email address by setting `SOCIALACCOUNT_EMAIL_VERIFICATION=True` and `SOCIALACCOUNT_EMAIL_REQUIRED=True`.

Baidu

The Baidu OAuth2 authentication documentation: http://developer.baidu.com/wiki/index.php?title=docs/oauth/refresh http://developer.baidu.com/wiki/index.php?title=docs/oauth/rest/file_data_apis_lista

Basecamp

App registration (get your key and secret here) https://integrate.37signals.com/

The Basecamp OAuth2 authentication documentation https://github.com/basecamp/api/blob/master/sections/authentication.md#oauth-2

Development callback URL https://localhost:8000/accounts/basecamp/login/callback/

Battle.net

The Battle.net OAuth2 authentication documentation https://dev.battle.net/docs/read/oauth

Register your app here (Mashery account required) https://dev.battle.net/apps/register

Development callback URL https://localhost:8000/accounts/battlenet/login/callback/

Note that in order to use battletags as usernames, you are expected to override either the `username` field on your User model, or to pass a custom validator which will accept the `#` character using the `ACCOUNT_USERNAME_VALIDATORS` setting. Such a validator is available in `socialaccount.providers.battlenet.validators.BattletagUsernameValidator`.

Bitbucket

App registration (get your key and secret here) https://bitbucket.org/account/user/{{yourusername}}/oauth-consumers/new

Make sure you select the Account:Read permission.

Development callback URL http://127.0.0.1:8000/accounts/bitbucket_oauth2/login/callback/

Note that Bitbucket calls the `client_id` *Key* in their user interface. Don't get confused by that; use the *Key* value for your `client_id` field.

Box

App registration (get your key and secret here) https://app.box.com/developers/services/edit/

Development callback URL http://localhost:8000/accounts/box/login/callback/

Dataporten

App registration (get your key and secret here) https://docs.dataporten.no/docs/gettingstarted/

Development callback URL http://localhost:8000/accounts/dataporten/login/callback

daum

App registration (get your key and secret here) https://developers.daum.net/console

Development callback URL http://127.0.0.1:8000/accounts/daum/login/callback/

DigitalOcean

App registration (get your key and secret here) https://cloud.digitalocean.com/settings/applications

Development callback URL http://127.0.0.1:8000/accounts/digitalocean/login/callback/

With the acquired access token you will have read permissions on the API by default. If you also need write access specify the scope as follows. See https://developers.digitalocean.com/documentation/oauth/#scopes for details.

```
SOCIALACCOUNT_PROVIDERS = {
    'digitalocean': {
        'SCOPE': [
            'read write',
        ],
    }
}
```

Discord

App registration and management (get your key and secret here) https://discordapp.com/developers/applications/me

Make sure to Add Redirect URI to your application.

Development callback (redirect) URL http://127.0.0.1:8000/accounts/discord/login/callback/

Doximity

Doximity OAuth2 implementation documentation https://www.doximity.com/developers/documentation#oauth

Request API keys here https://www.doximity.com/developers/api_signup

Development callback URL http://localhost:8000/accounts/doximity/login/callback/

Draugiem

App registration (get your key and secret here) https://www.draugiem.lv/applications/dev/create/?type=4

Authentication documentation https://www.draugiem.lv/applications/dev/docs/passport/

Development callback URL http://localhost:8000/accounts/draugiem/login/callback/

Dropbox

App registration (get your key and secret here) https://www.dropbox.com/developers/apps/

Development callback URL http://localhost:8000/accounts/dropbox/login/callback/

Note that Dropbox has deprecated version 1 of their API as of 28 June 2016. The original OAuth1 *dropbox* provider has been updated to OAuth2 and the newer URL endpoints for OAuth2 authentication and authorization.

The *dropbox_oauth2* provider is deprecated and will be removed after September 28, 2017.

Dwolla

App registration (get your key and secret here) https://dashboard-uat.dwolla.com/applications

Development callback URL http://127.0.0.1:8000/accounts/dwolla/login/callback/

```
SOCIALACCOUNT_PROVIDERS = {
    'dwolla': {
        'SCOPE': [
            'Send',
            'Transactions',
            'Funding',
            'AccountInfoFull',
        ],
        'ENVIROMENT':'sandbox',
    }
}
```

Edmodo

Edmodo OAuth2 documentation https://developers.edmodo.com/edmodo-connect/edmodo-connect-overview-getting-started/

You can optionally specify additional permissions to use. If no `SCOPE` value is set, the Edmodo provider will use `basic` by default:

```
SOCIALACCOUNT_PROVIDERS = {
    'edmodo': {
        'SCOPE': [
            'basic',
            'read_groups',
            'read_connections',
            'read_user_email',
            'create_messages',
            'write_library_items',
        ]
    }
}
```

Eve Online

Register your application at `https://developers.eveonline.com/applications/create`. Note that if you have `STORE_TOKENS` enabled (the default), you will need to set up your application to be able to request an OAuth scope. This means you will need to set it as having "CREST Access". The least obtrusive scope is "publicData".

Eventbrite

Log in and click your profile name in the top right navigation, then select `Account Settings`. Choose `App Management` near the bottom of the left navigation column. You can then click `Create A New App` on the upper left corner.

App registration https://www.eventbrite.com/myaccount/apps/

Fill in the form with the following link

Development callback URL http://127.0.0.1:8000/accounts/eventbrite/login/callback/

for both the `Application URL` and `OAuth Redirect URI`.

Evernote

Register your OAuth2 application at `https://dev.evernote.com/doc/articles/authentication.php`:

```
SOCIALACCOUNT_PROVIDERS = {
    'evernote': {
        'EVERNOTE_HOSTNAME': 'evernote.com'  # defaults to sandbox.evernote.com
    }
}
```

Facebook

For Facebook both OAuth2 and the Facebook Connect Javascript SDK are supported. You can even mix the two.

An advantage of the Javascript SDK may be a more streamlined user experience as you do not leave your site. Furthermore, you do not need to worry about tailoring the login dialog depending on whether or not you are using a mobile device. Yet, relying on Javascript may not be everybody's cup of tea.

To initiate a login use:

```
{% load socialaccount %}
{% providers_media_js %}
<a href="{% provider_login_url "facebook" method="js_sdk" %}">Facebook Connect</a>
```

or:

```
{% load socialaccount %}
<a href="{% provider_login_url "facebook" method="oauth2" %}">Facebook OAuth2</a>
```

The following Facebook settings are available:

```
SOCIALACCOUNT_PROVIDERS = {
    'facebook': {
        'METHOD': 'oauth2',
        'SCOPE': ['email', 'public_profile', 'user_friends'],
        'AUTH_PARAMS': {'auth_type': 'reauthenticate'},
        'INIT_PARAMS': {'cookie': True},
        'FIELDS': [
            'id',
            'email',
            'name',
            'first_name',
```

```
            'last_name',
            'verified',
            'locale',
            'timezone',
            'link',
            'gender',
            'updated_time',
        ],
        'EXCHANGE_TOKEN': True,
        'LOCALE_FUNC': 'path.to.callable',
        'VERIFIED_EMAIL': False,
        'VERSION': 'v2.5',
    }
}
```

METHOD: Either `js_sdk` or `oauth2`. The default is `oauth2`.

SCOPE: By default, the `email` scope is required depending on whether or not `SOCIALACCOUNT_QUERY_EMAIL` is enabled. Apps using permissions beyond `email`, `public_profile` and `user_friends` require review by Facebook. See Permissions with Facebook Login for more information.

AUTH_PARAMS: Use `AUTH_PARAMS` to pass along other parameters to the `FB.login` JS SDK call.

FIELDS: The fields to fetch from the Graph API `/me/?fields=` endpoint. For example, you could add the `'friends'` field in order to capture the user's friends that have also logged into your app using Facebook (requires `'user_friends'` scope).

EXCHANGE_TOKEN: The JS SDK returns a short-lived token suitable for client-side use. Set `EXCHANGE_TOKEN = True` to make a server-side request to upgrade to a long-lived token before storing in the `SocialToken` record. See Expiration and Extending Tokens.

LOCALE_FUNC: The locale for the JS SDK is chosen based on the current active language of the request, taking a best guess. This can be customized using the `LOCALE_FUNC` setting, which takes either a callable or a path to a callable. This callable must take exactly one argument, the request, and return a valid Facebook locale as a string, e.g. US English:

```
SOCIALACCOUNT_PROVIDERS = {
    'facebook': {
        'LOCALE_FUNC': lambda request: 'en_US'
    }
}
```

VERIFIED_EMAIL: It is not clear from the Facebook documentation whether or not the fact that the account is verified implies that the e-mail address is verified as well. For example, verification could also be done by phone or credit card. To be on the safe side, the default is to treat e-mail addresses from Facebook as unverified. But, if you feel that is too paranoid, then use this setting to mark them as verified. Due to lack of an official statement from the side of Facebook, attempts have been made to reverse engineer the meaning of the verified flag. Do know that by setting this to `True` you may be introducing a security risk.

VERSION: The Facebook Graph API version to use. The default is `v2.5`.

App registration (get your key and secret here) A key and secret key can be obtained by creating an app. After registration you will need to make it available to the public. In order to do that your app first has to be reviewed by Facebook.

Development callback URL Leave your App Domains empty and put `http://localhost:8000` in the section labeled `Website with Facebook Login`. Note that you'll need to add your site's actual domain to this section once it goes live.

Firefox Accounts

The Firefox Accounts provider is currently limited to Mozilla relying services but there is the intention, in the future, to allow third-party services to delegate authentication. There is no committed timeline for this.

The provider is OAuth2 based. More info: https://developer.mozilla.org/en-US/Firefox_Accounts

Note: This is not the same as the Mozilla Persona provider below.

The following Firefox Accounts settings are available:

```
SOCIALACCOUNT_PROVIDERS = {
    'fxa': {
        'SCOPE': ['profile'],
        'OAUTH_ENDPOINT': 'https://oauth.accounts.firefox.com/v1',
        'PROFILE_ENDPOINT': 'https://profile.accounts.firefox.com/v1',
    }
}
```

SCOPE: Requested OAuth2 scope. Default is ['profile'], which will work for applications on the Mozilla trusted whitelist. If your application is not on the whitelist, then define SCOPE to be ['profile:email', 'profile:uid'].

OAUTH_ENDPOINT: Explicitly set the OAuth2 endpoint. Default is the production endpoint "https://oauth.accounts.firefox.com/v1".

PROFILE_ENDPOINT: Explicitly set the profile endpoint. Default is the production endpoint and is "https://profile.accounts.firefox.com/v1".

Flickr

App registration (get your key and secret here) https://www.flickr.com/services/apps/create/

You can optionally specify the application permissions to use. If no `perms` value is set, the Flickr provider will use `read` by default.

```
SOCIALACCOUNT_PROVIDERS = {
    'flickr': {
        'AUTH_PARAMS': {
            'perms': 'write',
        }
    }
}
```

More info: https://www.flickr.com/services/api/auth.oauth.html#authorization

GitHub

App registration (get your key and secret here) https://github.com/settings/applications/new

Development callback URL http://127.0.0.1:8000/accounts/github/login/callback/

If you want more than just read-only access to public data, specify the scope as follows. See https://developer.github.com/v3/oauth/#scopes for details.

```
SOCIALACCOUNT_PROVIDERS = {
    'github': {
        'SCOPE': [
            'user',
```

```
            'repo',
            'read:org',
        ],
    }
}
```

Enterprise Support

If you use GitHub Enterprise add your server URL to your Django settings as follows:

```
SOCIALACCOUNT_PROVIDERS = {
    'github': {
        'GITHUB_URL': 'https://your.github-server.domain',
    }
}
```

GitLab

The GitLab provider works by default with https://gitlab.com. It allows you to connect to your private GitLab server and use GitLab as an OAuth2 authentication provider as described in GitLab docs at http://doc.gitlab.com/ce/integration/oauth_provider.html

The following GitLab settings are available, if unset https://gitlab.com will be used.

GITLAB_URL: Override endpoint to request an authorization and access token. For your private GitLab server you use: `https://your.gitlab.server.tld`

Example:

```
SOCIALACCOUNT_PROVIDERS = {
    'gitlab': {
        'GITLAB_URL': 'https://your.gitlab.server.tld',
    }
}
```

Google

The Google provider is OAuth2 based.

More info: http://code.google.com/apis/accounts/docs/OAuth2.html#Registering

App registration

Create a google app to obtain a key and secret through the developer console.

Google Developer Console https://console.developers.google.com/

After you create a project you will have to create a "Client ID" and fill in some project details for the consent form that will be presented to the client.

Under "APIs & auth" go to "Credentials" and create a new Client ID. Probably you will want a "Web application" Client ID. Provide your domain name or test domain name in "Authorized JavaScript origins". Finally fill in `http://127.0.0.1:8000/accounts/google/login/callback/` in the "Authorized redirect URI" field. You can

fill multiple URLs, one for each test domain. After creating the Client ID you will find all details for the Django configuration on this page.

Users that login using the app will be presented a consent form. For this to work additional information is required. Under "APIs & auth" go to "Consent screen" and at least provide an email and product name.

Django configuration

The app credentials are configured for your Django installation via the admin interface. Create a new socialapp through `/admin/socialaccount/socialapp/`.

Fill in the form as follows:

- Provider, "Google"
- Name, your pick, suggest "Google"
- Client id, is called "Client ID" by Google
- Secret key, is called "Client secret" by Google
- Key, is not needed, leave blank.

Optionally, you can specify the scope to use as follows:

```
SOCIALACCOUNT_PROVIDERS = {
    'google': {
        'SCOPE': [
            'profile',
            'email',
        ],
        'AUTH_PARAMS': {
            'access_type': 'online',
        }
    }
}
```

By default, `profile` scope is required, and optionally `email` scope depending on whether or not `SOCIALACCOUNT_QUERY_EMAIL` is enabled.

Instagram

App registration (get your key and secret here) https://www.instagram.com/developer/clients/manage/

Development callback URL http://localhost:8000/accounts/instagram/login/callback/

Kakao

App registration (get your key here) https://developers.kakao.com/apps

Development callback URL http://localhost:8000/accounts/kakao/login/callback/

Line

App registration (get your key and secret here) https://business.line.me

Development callback URL http://127.0.0.1:8000/accounts/line/login/callback/

LinkedIn

The LinkedIn provider comes in two flavors: OAuth 1.0 (`allauth.socialaccount.providers.linkedin`) and OAuth 2.0 (`allauth.socialaccount.providers.linkedin_oauth2`).

You can specify the scope and fields to fetch as follows:

```
SOCIALACCOUNT_PROVIDERS = {
    'linkedin': {
        'SCOPE': [
            'r_emailaddress',
        ],
        'PROFILE_FIELDS': [
            'id',
            'first-name',
            'last-name',
            'email-address',
            'picture-url',
            'public-profile-url',
        ]
    }
}
```

By default, `r_emailaddress` scope is required depending on whether or not `SOCIALACCOUNT_QUERY_EMAIL` is enabled.

Note: if you are experiencing issues where it seems as if the scope has no effect you may be using an old LinkedIn app that is not scope enabled. Please refer to `https://developer.linkedin.com/forum/when-will-old-apps-have-scope-parameter-enabled` for more background information.

Furthermore, we have experienced trouble upgrading from OAuth 1.0 to OAuth 2.0 using the same app. Attempting to do so resulted in a weird error message when fetching the access token:

```
missing required parameters, includes an invalid parameter value, parameter more then␣
↪once. : Unable to retrieve access token : authorization code not found
```

App registration (get your key and secret here) https://www.linkedin.com/secure/developer?newapp=

Authorized Redirect URLs (OAuth2)

Add any you need (up to 200) consisting of:

{ACCOUNT_DEFAULT_HTTP_PROTOCOL}://{hostname}{:optional_port}/{allauth_base_url}/linkedin_oauth2/login/callback/

For example when using the built-in django server and default settings:

http://localhost:8000/accounts/linkedin_oauth2/login/callback/

Development "Accept" and "Cancel" redirect URL (OAuth 1.0a)

Leave the OAuth1 redirect URLs empty.

MailChimp (OAuth2)

MailChimp has a simple API for working with your own data and a good library already exists for this use. However, to allow other MailChimp users to use an app you develop, the OAuth2 API allows those users to give or revoke access without creating a key themselves.

Registering a new app

Instructions for generating your own OAuth2 app can be found at https://developer.mailchimp.com/documentation/mailchimp/guides/how-to-use-oauth2/. It is worth reading that carefully before following the instructions below.

Login via https://login.mailchimp.com/, which will redirect you to `https://usX.admin.mailchimp.com/` where the prefix `usX` (`X` is an integer) is the subdomain you need to connect to. Click on your username in the top right corner and select *Profile*. On the next page select *Extras* then click API keys, which should lead you to:

App registration (where X is dependent on your account) https://usX.admin.mailchimp.com/account/oauth2/

Fill in the form with the following URL for local development:

Development callback URL https://127.0.0.1:8000/accounts/mailchimp/login/callback/

Testing Locally

Note the requirement of **https**. If you would like to test OAuth2 authentication locally before deploying a default django project will raise errors because development mode does not support `https`. One means of circumventing this is to install `django-extensions`:

```
pip install django-extensions
```

add it to your `INSTALLED_APPS`

```
INSTALLED_APPS = (
    ...
    'django_extensions',
    ...
)
```

and then run:

```
./manage.py runserver_plus --cert cert
```

which should allow you to test locally via https://127.0.0.1:8000. Some browsers may require enabling this on localhost and not support by default and ask for permission.

Naver

App registration (get your key and secret here) https://developers.naver.com/appinfo

Development callback URL http://localhost:8000/accounts/naver/login/callback/

Odnoklassniki

App registration (get your key and secret here) http://apiok.ru/wiki/pages/viewpage.action?pageId=42476486

Development callback URL http://example.com/accounts/odnoklassniki/login/callback/

OpenID

The OpenID provider does not require any settings per se. However, a typical OpenID login page presents the user with a predefined list of OpenID providers and allows the user to input their own OpenID identity URL in case their provider is not listed by default. The list of providers displayed by the builtin templates can be configured as follows:

```
SOCIALACCOUNT_PROVIDERS = {
    'openid': {
        'SERVERS': [
            dict(id='yahoo',
                 name='Yahoo',
                 openid_url='http://me.yahoo.com'),
            dict(id='hyves',
                 name='Hyves',
                 openid_url='http://hyves.nl'),
            dict(id='google',
                 name='Google',
                 openid_url='https://www.google.com/accounts/o8/id'),
        ]
    }
}
```

You can manually specify extra_data you want to request from server as follows:

```
SOCIALACCOUNT_PROVIDERS = \
    { 'openid':
        { 'SERVERS':
             [ dict(id='mojeid',
                  name='MojeId',
                  openid_url='https://mojeid.cz/endpoint/',
                  extra_attributes = [
                      ('phone', 'http://axschema.org/contact/phone/default', False),
                      ('birth_date', 'http://axschema.org/birthDate', False,),
                  ])]}}
```

Attributes are in form (id, name, required) where id is key in extra_data field of socialaccount, name is identifier of requested attribute and required specifies whether attribute is required.

If you want to manually include login links yourself, you can use the following template tag:

```
{% load socialaccount %}
<a href="{% provider_login_url "openid" openid="https://www.google.com/accounts/o8/id
↪" next="/success/url/" %}">Google</a>
```

ORCID

The ORCID provider should work out of the box provided that you are using the Production ORCID registry and the public API. In other settings, you will need to define the API you are using in your site's settings, as follows:

```
SOCIALACCOUNT_PROVIDERS = {
    'orcid': {
        # Base domain of the API. Default value: 'orcid.org', for the production API
        'BASE_DOMAIN':'sandbox.orcid.org',  # for the sandbox API
        # Member API or Public API? Default: False (for the public API)
        'MEMBER_API': True,  # for the member API
    }
}
```

Patreon

App registration (get your key and secret here) https://www.patreon.com/platform/documentation/clients

Development callback URL http://127.0.0.1:8000/accounts/patreon/login/callback/

Paypal

The following Paypal settings are available:

```
SOCIALACCOUNT_PROVIDERS = {
    'paypal': {
        'SCOPE': ['openid', 'email'],
        'MODE': 'live',
    }
}
```

SCOPE: In the Paypal developer site, you must also check the required attributes for your application. For a full list of scope options, see https://developer.paypal.com/docs/integration/direct/identity/attributes/

MODE: Either `live` or `test`. Set to test to use the Paypal sandbox.

App registration (get your key and secret here) https://developer.paypal.com/webapps/developer/applications/myapps

Development callback URL http://example.com/accounts/paypal/login/callback

Persona

Note: Mozilla Persona was shut down on November 30th 2016. See the announcement for details.

Mozilla Persona requires one setting, the "AUDIENCE" which needs to be the hardcoded hostname and port of your website. See https://developer.mozilla.org/en-US/Persona/Security_Considerations#Explicitly_specify_the_audience_parameter for more information why this needs to be set explicitly and can't be derived from user provided data:

```
SOCIALACCOUNT_PROVIDERS = {
    'persona': {
        'AUDIENCE': 'https://www.example.com',
    }
}
```

The optional `REQUEST_PARAMETERS` dictionary contains parameters that are passed as is to the `navigator.id.request()` method to influence the look and feel of the Persona dialog:

```
SOCIALACCOUNT_PROVIDERS = {
    'persona': {
        'AUDIENCE': 'https://www.example.com',
        'REQUEST_PARAMETERS': {'siteName': 'Example'},
    }
}
```

Pinterest

The Pinterest OAuth2 documentation:

https://developers.pinterest.com/docs/api/overview/#authentication

You can optionally specify additional permissions to use. If no `SCOPE` value is set, the Pinterest provider will use `read_public` by default.

```
SOCIALACCOUNT_PROVIDERS = {
    'pinterest': {
        'SCOPE': [
            'read_public',
            'read_relationships',
        ]
    }
}
```

SCOPE: For a full list of scope options, see https://developers.pinterest.com/docs/api/overview/#scopes

Reddit

App registration (get your key and secret here) https://www.reddit.com/prefs/apps/

Development callback URL http://localhost:8000/accounts/reddit/login/callback/

By default, access to Reddit is temporary. You can specify the `duration` auth parameter to make it `permanent`.

You can optionally specify additional permissions to use. If no `SCOPE` value is set, the Reddit provider will use `identity` by default.

In addition, you should override your user agent to comply with Reddit's API rules, and specify something in the format `<platform>:<app ID>:<version string> (by /u/<reddit username>)`. Otherwise, you will risk additional rate limiting in your application.

```
SOCIALACCOUNT_PROVIDERS = {
    'reddit': {
        'AUTH_PARAMS': {'duration': 'permanent'},
        'SCOPE': ['identity', 'submit'],
        'USER_AGENT': 'django:myappid:1.0 (by /u/yourredditname)',
    }
}
```

Shopify

The Shopify provider requires a `shop` parameter to login. For example, for a shop `petstore.myshopify.com`, use this:

```
/accounts/shopify/login/?shop=petstore
```

You can create login URLs like these as follows:

```
{% provider_login_url "shopify" shop="petstore" %}
```

For setting up authentication in your app, use this url as your `App URL` (if your server runs at localhost:8000):

```
http://localhost:8000/accounts/shopify/login/
```

And set `Redirection URL` to:

```
http://localhost:8000/accounts/shopify/login/callback/
```

Embedded Apps

If your Shopify app is embedded you will want to tell allauth to do the required JS (rather than server) redirect.:

```
SOCIALACCOUNT_PROVIDERS = {
    'shopify': {
        'IS_EMBEDDED': True,
    }
}
```

Note that there is more an embedded app creator must do in order to have a page work as an iFrame within Shopify (building the x_frame_exempt landing page, handing session expiration, etc...). However that functionality is outside the scope of django-allauth.

Online/per-user access mode Shopify has two access modes, offline (the default) and online/per-user. Enabling 'online' access will cause all-auth to tie the logged in Shopify user to the all-auth account (rather than the shop as a whole).:

```
SOCIALACCOUNT_PROVIDERS = {
    'shopify': {
        'AUTH_PARAMS': {'grant_options[]': 'per-user'},
    }
}
```

Slack

App registration (get your key and secret here) https://api.slack.com/apps/new

Development callback URL http://example.com/accounts/slack/login/callback/

API documentation https://api.slack.com/docs/sign-in-with-slack

SoundCloud

SoundCloud allows you to choose between OAuth1 and OAuth2. Choose the latter.

App registration (get your key and secret here) http://soundcloud.com/you/apps/new

Development callback URL http://example.com/accounts/soundcloud/login/callback/

Stack Exchange

Register your OAuth2 app over at `http://stackapps.com/apps/oauth/register`. Do not enable "Client Side Flow". For local development you can simply use "localhost" for the OAuth domain.

As for all providers, provider specific data is stored in `SocialAccount.extra_data`. For Stack Exchange we need to choose what data to store there by choosing the Stack Exchange site (e.g. Stack Overflow, or Server Fault). This can be controlled by means of the `SITE` setting:

```
SOCIALACCOUNT_PROVIDERS = {
    'stackexchange': {
        'SITE': 'stackoverflow',
    }
}
```

Stripe

You can register your OAuth2 app via the admin interface http://example.com/accounts/stripe/login/callback/

See more in documentation https://stripe.com/docs/connect/standalone-accounts

Twitch

App registration (get your key and secret here) http://www.twitch.tv/kraken/oauth2/clients/new

Twitter

You will need to create a Twitter app and configure the Twitter provider for your Django application via the admin interface.

App registration

To register an app on Twitter you will need a Twitter account. With an account, you can create a new app via:

```
https://apps.twitter.com/app/new
```

In the app creation form fill in the development callback URL:

```
http://127.0.0.1:8000/accounts/twitter/login/callback/
```

Twitter won't allow using http://localhost:8000.

For production use a callback URL such as:

```
http://{{yourdomain}}.com/accounts/twitter/login/callback/
```

To allow users to login without authorizing each session, select "Allow this application to be used to Sign in with Twitter" under the application's "Settings" tab.

App database configuration through admin

The second part of setting up the Twitter provider requires you to configure your Django application. Configuration is done by creating a Socialapp object in the admin. Add a social app on the admin page:

```
/admin/socialaccount/socialapp/
```

Use the twitter keys tab of your application to fill in the form. It's located:

```
https://apps.twitter.com/app/{{yourappid}}/keys
```

The configuration is as follows:

- Provider, "Twitter"
- Name, your pick, suggest "Twitter"
- Client id, is called "Consumer Key (API Key)" on Twitter
- Secret key, is called "Consumer Secret (API Secret)" on Twitter
- Key, is not needed, leave blank

Untappd

App registration

https://untappd.com/api/register?register=new

In the app creation form fill in the development callback URL, e.g.:

```
http://127.0.0.1:8000/accounts/untappd/login/callback/
```

For production, make it your production host, e.g.:

```
http://yoursite.com/accounts/untappd/login/callback/
```

SocialApp configuration

The configuration values come from your API dashboard on Untappd:

https://untappd.com/api/dashboard

- Provider: "Untappd"
- Name: "Untappd"
- Client id: "Client ID" from Untappd
- Secret key: "Client Secret" from Untappd
- Sites: choose your site

Vimeo

App registration (get your key and secret here) https://developer.vimeo.com/apps

Development callback URL http://localhost:8000

VK

App registration https://vk.com/editapp?act=create

Development callback URL ("Site address") http://localhost

Windows Live

The Windows Live provider currently does not use any settings in `SOCIALACCOUNT_PROVIDERS`.

App registration (get your key and secret here) https://apps.dev.microsoft.com/#/appList

Development callback URL http://localhost:8000/accounts/windowslive/login/callback/

Microsoft calls the "client_id" an "Application Id" and it is a UUID. Also, the "client_secret" is not created by default, you must edit the application after it is created, then click "Generate New Password" to create it.

Weibo

Register your OAuth2 app over at `http://open.weibo.com/apps`. Unfortunately, Weibo does not allow for specifying a port number in the authorization callback URL. So for development purposes you have to use a callback url of the form `http://127.0.0.1/accounts/weibo/login/callback/` and run `runserver 127.0.0.1:80`.

Weixin

The Weixin OAuth2 documentation:

> https://open.weixin.qq.com/cgi-bin/showdocument?action=dir_list&t=resource/res_list&verify=1&id=open1419316505&token=&lang=zh_CN

Weixin supports two kinds of oauth2 authorization, one for open platform and one for media platform, AUTHORIZE_URL is the only difference between them, you can specify `AUTHORIZE_URL` in setting, If no `AUTHORIZE_URL` value is set will support open platform by default, which value is `https://open.weixin.qq.com/connect/qrconnect`.

You can optionally specify additional scope to use. If no `SCOPE` value is set, will use `snsapi_login` by default.

```
SOCIALACCOUNT_PROVIDERS = {
    'weixin': {
        'AUTHORIZE_URL': 'https://open.weixin.qq.com/connect/oauth2/authorize',  #␣
↪for media platform
    }
}
```

Xing

App registration (get your key and secret here) https://dev.xing.com/applications

Development callback URL http://localhost:8000

Signals

There are several signals emitted during authentication flows. You can hook to them for your own needs.

allauth.account

- `allauth.account.signals.user_logged_in(request, user)`

  Sent when a user logs in.

- `allauth.account.signals.user_logged_out(request, user)`

  Sent when a user logs out.

- `allauth.account.signals.user_signed_up(request, user)`

  Sent when a user signs up for an account. This signal is typically followed by a `user_logged_in`, unless e-mail verification prohibits the user to log in.

- `allauth.account.signals.password_set(request, user)`

  Sent when a password has been successfully set for the first time.

- `allauth.account.signals.password_changed(request, user)`

  Sent when a password has been successfully changed.

- `allauth.account.signals.password_reset(request, user)`

  Sent when a password has been successfully reset.

- `allauth.account.signals.email_confirmed(request, email_address)`

  Sent after the email address in the db was updated and set to confirmed.

- `allauth.account.signals.email_confirmation_sent(request, confirmation, signup)`

  Sent right after the email confirmation is sent.

- `allauth.account.signals.email_changed(request, user, from_email_address, to_email_address)`

  Sent when a primary email address has been changed.

- `allauth.account.signals.email_added(request, user, email_address)`

  Sent when a new email address has been added.

- `allauth.account.signals.email_removed(request, user, email_address)`

  Sent when an email address has been deleted.

allauth.socialaccount

- `allauth.socialaccount.signals.pre_social_login(request, sociallogin)`

  Sent after a user successfully authenticates via a social provider, but before the login is fully processed. This signal is emitted as part of the social login and/or signup process, as well as when connecting additional social accounts to an existing account. Access tokens and profile information, if applicable for the provider, is provided.

- `allauth.socialaccount.signals.social_account_added(request, sociallogin)`

  Sent after a user connects a social account to a their local account.

- `allauth.socialaccount.signals.social_account_updated(request, sociallogin)`

  Sent after a social account has been updated. This happens when a user logs in using an already connected social account, or completes a *connect* flow for an already connected social account. Useful if you need to unpack extra data for social accounts as they are updated.

- `allauth.socialaccount.signals.social_account_removed(request, socialaccount)`

  Sent after a user disconnects a social account from their local account.

Views

Login (account_login)

Users login via the allauth.account.views.LoginView view over at /accounts/login/ (URL name account_login). When users attempt to login while their account is inactive (user.is_active) they are presented with the account/account_inactive.html template.

Signup (account_signup)

Users sign up via the allauth.account.views.SignupView view over at /accounts/signup/ (URL name account_signup).

Logout (account_logout)

The logout view (allauth.account.views.LogoutView) over at /accounts/logout/ (URL name account_logout) requests for confirmation before logging out. The user is logged out only when the confirmation is received by means of a POST request.

If you are wondering why, consider what happens when a malicious user embeds the following image in a post:

```
<img src="http://example.com/accounts/logout/">
```

For this and more background information on the subject, see:

- https://code.djangoproject.com/ticket/15619
- http://stackoverflow.com/questions/3521290/logout-get-or-post

If you insist on having logout on GET, then please consider adding a bit of Javascript to automatically turn a click on a logout link into a POST. As a last resort, you can set ACCOUNT_LOGOUT_ON_GET to True.

Password Management

Authenticated users can manage their password account using the allauth.account.views.PasswordSetView and allauth.account.views.PasswordChangeView views, over at /accounts/password/set/ respectively /accounts/password/change/ (URL names account_set_password and account_change_password respectively).

Users are redirected between these views, according to whether or not they have setup a password (user.has_usable_password()). Typically, when users signup via a social provider they will not have a password set.

Password Reset (account_reset_password)

Users can request a password reset using the allauth.account.views.PasswordResetView view over at /accounts/password/reset/ (URL name account_reset_password). An e-mail will be sent containing a reset link pointing to PasswordResetFromKeyView view.

E-mails Management (account_email)

Users manage the e-mail addresses tied to their account using the allauth.account.views.EmailView view over at /accounts/email/ (URL name account_email). Here, users can add (and verify) e-mail addresses, remove e-mail addresses, and choose a new primary e-mail address.

E-mail Verification

Depending on the setting ACCOUNT_EMAIL_VERIFICATION, a verification e-mail is sent pointing to the allauth.account.views.ConfirmEmailView view.

The setting ACCOUNT_CONFIRM_EMAIL_ON_GET determines whether users have to manually confirm the address by submiting a confirmation form, or whether the address is automatically confirmed by a mere GET request.

Social Connections (socialaccount_connections)

The allauth.socialaccount.views.ConnectionsView view over at /accounts/social/connections/ (URL name socialaccount_connections) allows users to manage the social accounts tied to their local account.

Templates

Overridable templates

allauth ships many templates, viewable in the allauth/templates directory.

For instance, the view corresponding to the account_login URL uses the template account/login.html. If you create a file with this name in your code layout, it can override the one shipped with allauth.

Template Tags

The following template tag libraries are available:

- account: tags for dealing with accounts in general
- socialaccount: tags focused on social accounts

Account Tags

Use user_display to render a user name without making assumptions on how the user is represented (e.g. render the username, or first name?):

```
{% load account %}

{% user_display user %}
```

Or, if you need to use in a {% blocktrans %}:

```
{% load account %}

{% user_display user as user_display %}
{% blocktrans %}{{ user_display }} has logged in...{% endblocktrans %}
```

Then, override the `ACCOUNT_USER_DISPLAY` setting with your project specific user display callable.

Social Account Tags

Use the `provider_login_url` tag to generate provider specific login URLs:

```
{% load socialaccount %}

<a href="{% provider_login_url "openid" openid="https://www.google.com/accounts/o8/id
↪" next="/success/url/" %}">Google</a>
<a href="{% provider_login_url "twitter" %}">Twitter</a>
```

Here, you can pass along an optional `process` parameter that indicates how to process the social login. You can choose between `login` and `connect`:

```
<a href="{% provider_login_url "twitter" process="connect" %}">Connect a Twitter␣
↪account</a>
```

Furthermore, you can pass along an `action` parameter with value `reauthenticate` to indicate that you want the user to be re-prompted for authentication even if they already signed in before. For now, this is supported by Facebook, Google and Twitter only.

For Javascript based logins (e.g. when you enable the Facebook JS SDK), you will need to make sure that the required Javascript is loaded. The following tag loads all scripts for the enabled providers:

```
{% providers_media_js %}
```

For easy access to the social accounts for a user use:

```
{% get_social_accounts user as accounts %}
```

Then:

```
{{accounts.twitter}} -- a list of connected Twitter accounts
{{accounts.twitter.0}} -- the first Twitter account
{% if accounts %} -- if there is at least one social account
```

Finally, social authentication providers configured for the current site can be retrieved via:

```
{% get_providers as socialaccount_providers %}
```

Which will populate the `socialaccount_providers` variable in the template context with a list of configured social authentication providers. This supersedes the context processor used in version 0.21 and below.

Decorators

Verified E-mail Required

Even when email verification is not mandatory during signup, there may be circumstances during which you really want to prevent unverified users to proceed. For this purpose you can use the following decorator:

```
from allauth.account.decorators import verified_email_required

@verified_email_required
def verified_users_only_view(request):
    ...
```

The behavior is as follows:

- If the user isn't logged in, it acts identical to the `login_required` decorator.
- If the user is logged in but has no verified e-mail address, an e-mail verification mail is automatically resent and the user is presented with a page informing them they need to verify their email address.

Advanced Usage

HTTPS

This app currently provides no functionality for enforcing views to be HTTPS only, or switching from HTTP to HTTPS (and back) on demand. There are third party packages aimed at providing precisely this, please use these .

What is provided is the following:

- The protocol to be used for generating links (e.g. password forgotten) for e-mails is configurable by means of the `ACCOUNT_DEFAULT_HTTP_PROTOCOL` setting.
- Automatically switching to HTTPS is built-in for OAuth providers that require this (e.g. Amazon). However, remembering the original protocol before the switch and switching back after the login is not provided.

Custom User Models

If you use a custom user model you need to specify what field represents the `username`, if any. Here, `username` really refers to the field representing the nick name the user uses to login, and not some unique identifier (possibly including an e-mail address) as is the case for Django's `AbstractBaseUser.USERNAME_FIELD`.

Meaning, if your custom user model does not have a `username` field (again, not to be mistaken with an e-mail address or user id), you will need to set `ACCOUNT_USER_MODEL_USERNAME_FIELD` to `None`. This will disable username related functionality in `allauth`. Remember to also to set `ACCOUNT_USERNAME_REQUIRED` to `False`.

Similarly, you will need to set `ACCOUNT_USER_MODEL_EMAIL_FIELD` to `None`, or the proper field (if other than `email`).

For example, if you want to use a custom user model that has `email` as the identifying field, and you don't want to collect usernames, you need the following in your settings.py:

```
ACCOUNT_USER_MODEL_USERNAME_FIELD = None
ACCOUNT_EMAIL_REQUIRED = True
ACCOUNT_USERNAME_REQUIRED = False
ACCOUNT_AUTHENTICATION_METHOD = 'email'
```

Creating and Populating User instances

The following adapter methods can be used to intervene in how User instances are created, and populated with data

- `allauth.account.adapter.DefaultAccountAdapter`:

  - `is_open_for_signup(self, request)`: The default function returns `True`. You can override this method by returning `False` if you want to disable account signup.
  - `new_user(self, request)`: Instantiates a new, empty `User`.
  - `save_user(self, request, user, form)`: Populates and saves the `User` instance using information provided in the signup form.
  - `populate_username(self, request, user)`: Fills in a valid username, if required and missing. If the username is already present it is assumed to be valid (unique).
  - `confirm_email(self, request, email_address)`: Marks the email address as confirmed and saves to the db.
  - `generate_unique_username(self, txts, regex=None)`: Returns a unique username from the combination of strings present in txts iterable. A regex pattern can be passed to the method to make sure the generated username matches it.
- `allauth.socialaccount.adapter.DefaultSocialAccountAdapter`:
  - `is_open_for_signup(self, request)`: The default function returns that is the same as `ACCOUNT_ADAPTER` in `settings.py`. You can override this method by returning `True`/`False` if you want to enable/disable socialaccount signup.
  - `new_user(self, request, sociallogin)`: Instantiates a new, empty `User`.
  - `save_user(self, request, sociallogin, form=None)`: Populates and saves the `User` instance (and related social login data). The signup form is not available in case of auto signup.
  - `populate_user(self, request, sociallogin, data)`: Hook that can be used to further populate the user instance (`sociallogin.account.user`). Here, `data` is a dictionary of common user properties (`first_name`, `last_name`, `email`, `username`, `name`) that the provider already extracted for you.

Invitations

Invitation handling is not supported, and most likely will not be any time soon. An invitation app could cover anything ranging from invitations of new users, to invitations of existing users to participate in restricted parts of the site. All in all, the scope of invitation handling is large enough to warrant being addressed in an app of its own.

Still, everything is in place to easily hook up any third party invitation app. The account adapter (`allauth.account.adapter.DefaultAccountAdapter`) offers the following methods:

- `is_open_for_signup(self, request)`. You can override this method to, for example, inspect the session to check if an invitation was accepted.
- `stash_verified_email(self, request, email)`. If an invitation was accepted by following a link in a mail, then there is no need to send e-mail verification mails after the signup is completed. Use this method to record the fact that an e-mail address was verified.

Sending E-mail

E-mails sent (e.g. in case of password forgotten, or e-mail confirmation) can be altered by providing your own templates. Templates are named as follows:

```
account/email/email_confirmation_subject.txt
account/email/email_confirmation_message.txt
```

In case you want to include an HTML representation, add an HTML template as follows:

```
account/email/email_confirmation_message.html
```

The project does not contain any HTML email templates out of the box. When you do provide these yourself, note that both the text and HTML versions of the message are sent.

If this does not suit your needs, you can hook up your own custom mechanism by overriding the `send_mail` method of the account adapter (`allauth.account.adapter.DefaultAccountAdapter`).

Custom Redirects

If redirecting to statically configurable URLs (as specified in your project settings) is not flexible enough, then you can override the following adapter methods:

- `allauth.account.adapter.DefaultAccountAdapter`:
  - `get_login_redirect_url(self, request)`
  - `get_logout_redirect_url(self, request)`
  - `get_email_confirmation_redirect_url(self, request)`
- `allauth.socialaccount.adapter.DefaultSocialAccountAdapter`:
  - `get_connect_redirect_url(self, request, socialaccount)`

For example, redirecting to `/accounts/<username>/` can be implemented as follows:

```
# project/settings.py:
ACCOUNT_ADAPTER = 'project.users.adapter.MyAccountAdapter'

# project/users/adapter.py:
from django.conf import settings
from allauth.account.adapter import DefaultAccountAdapter

class MyAccountAdapter(DefaultAccountAdapter):

    def get_login_redirect_url(self, request):
        path = "/accounts/{username}/"
        return path.format(username=request.user.username)
```

Messages

The Django messages framework (`django.contrib.messages`) is used if it is listed in `settings.INSTALLED_APPS`. All messages (as in `django.contrib.messages`) are configurable by overriding their respective template. If you want to disable a message simply override the message template with a blank one.

Admin

The Django admin site (`django.contrib.admin`) does not use Django allauth by default. Since Django admin provides a custom login view, it does not go through the normal Django allauth workflow.

Warning: This limitation means that Django allauth features are not applied to the Django admin site:

- `ACCOUNT_LOGIN_ATTEMPTS_LIMIT` and `ACCOUNT_LOGIN_ATTEMPTS_TIMEOUT` do not protect Django's admin login from being brute forced.

- Any other custom workflow that overrides the Django allauth adapter's login method will not be applied.

An easy workaround for this is to require users to login before going to the Django admin site's login page (note that following would need to be applied to every instance of `AdminSite`):

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

admin.site.login = login_required(admin.site.login)
```

Customizing providers

When an existing provider doesn't quite meet your needs, you might find yourself needing to customize a provider.

This can be achieved by subclassing an existing provider and making your changes there. Providers are defined as django applications, so typically customizing one will mean creating a django application in your project, containing your customized urls.py, views.py and provider.py files. What behaviour you can customize is beyond the scope of this documentation.

Warning: In your `provider.py` file, you will need to expose the provider class by having a module level attribute called `provider_classes` with your custom classes in a list. This allows your custom provider to be registered properly on the basis of the `INSTALLED_APPS` setting.

Be sure to use a custom id property on your provider class such that its default URLs do not clash with the provider you are subclassing.

```
class GoogleNoDefaultScopeProvider(GoogleProvider):
    id = 'google_no_scope'

    def get_default_scope(self):
        return []

provider_classes = [GoogleNoDefaultScopeProvider]
```

Frequently Asked Questions

Overall

Why don't you implement support for ... ?

This app is just about authentication. Anything that is project specific, such as making choices on what to display in a profile page, or, what information is stored for a user (e.g. home address, or favorite color?), is beyond scope and therefore not offered.

This information is nice and all, but... I need more!

Here are a few third party resources to help you get started:

- https://speakerdeck.com/tedtieken/signing-up-and-signing-in-users-in-django-with-django-allauth

- https://stackoverflow.com/questions/tagged/django-allauth
- http://www.sarahhagstrom.com/2013/09/the-missing-django-allauth-tutorial/
- https://github.com/aellerton/demo-allauth-bootstrap

I think I found a security issue... now what?

Please report security issues only to django-allauth-security@googlegroups.com. This is a private list only open to long-time, highly trusted django-allauth developers, and its archives are not public.

You may also want to subscribe to django-allauth-announce@googlegroups.com to get notified about security releases.

Troubleshooting

The /accounts/ URL is giving me a 404

There is no such URL. Try `/accounts/login/` instead.

When I attempt to login I run into a 404 on /accounts/profile/

When you end up here you have successfully logged in. However, you will need to implement a view for this URL yourself, as whatever is to be displayed here is project specific. You can also decide to redirect elsewhere:

https://docs.djangoproject.com/en/dev/ref/settings/#login-redirect-url

When I sign up I run into connectivity errors (connection refused et al)

You probably have not got an e-mail (SMTP) server running on the machine you are developing on. Therefore, `allauth` is unable to send verification mails.

You can work around this by adding the following line to `settings.py`:

EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'

This will avoid the need for an SMTP server as e-mails will be printed to the console. For more information, please refer to:

https://docs.djangoproject.com/en/dev/ref/settings/#email-host

Why did you just close my issue?

Time is limited and I have to pick my battles. Please do not file the following types of issues:

- **Support questions, installation instructions, "How do I...?":** please direct these questions elsewhere, for example here: https://stackoverflow.com/questions/tagged/django-allauth
- **Documentation related complaints:** while the documentation can most certainly be improved, I am adhering to the debatable principle that keeping open issues around with respect to documentation is not very helpful in improving things. Please step in and file a pull request if you feel there is something unclear.
- **Project specific integration trouble:** In cases where `allauth` is just one piece of the puzzle and for example a stack trace indicates another module crashing, please try to come up stripped version of the issue where it is clear that `allauth` is the one that is misbehaving.

Release Notes

0.33.0 (2017-08-20)

Note worthy changes

- Security: password reset tokens are now prevented from being leaked through the password reset URL.
- New providers: Patreon, Authentiq, Dataporten.
- Dropbox has been upgraded to API V2.
- New translation: Norwegian.

Backwards incompatible changes

- Dropped support for Django 1.9.

0.32.0 (2017-04-27)

Note worthy changes

- Improved AJAX support: the account management views (change/set password, manage e-mail addresses and social connections) now support AJAX GET requests. These views hand over all the required data for you to build your frontend application upon.
- New providers: Dwolla, Trello.
- Shopify: support for per-user access mode.

Backwards incompatible changes

- In previous versions, the views only responded with JSON responses when issuing AJAX requests of type POST. Now, the views also respond in JSON when making AJAX GET requests.
- The structure of the response for AJAX requests has changed. Previously, it contained a `form_errors` key containing all form validation errors, if any. Now, it contains a `form` key that describes the complete form, including the fields. Field specific errors are placed in `form.fields['some_field'].errors`, non-field errors in `form.errors`.
- The parameters passed to the Facebook JS SDK `FB.init()` method used to contain `cookie`, `status`, and `xfbml`, all set to `true`. These parameters are no longer explicitly passed. You can use the newly introduced `INIT_PARAMS` provider setting to provide your own values.

0.31.0 (2017-02-28)

Note worthy changes

- Added a new `user_logged_out` signal.
- OpenId: Added support for requesting additional data.
- New providers: Auth0, Box, Line, Naver, Kakao, Daum, MailChimp, Eventbrite.

Backwards incompatible changes

- Django 1.7 / Python 3.2 compatibility has been dropped.
- Due to providers being registered in the same file as their definition it was impossible to subclass a provider without having the parent be registered. This has been addressed. If you have implemented a custom provider, you will need to change `providers.registry.register(CustomProvider)` into `provider_classes = [CustomProvider]`.

0.30.0 (2017-01-01)

Note worthy changes

- Changed the algorithm that generates unique usernames. Previously, in case the provider did not hand over any information to base the username on, the username "user" extended with an ever increasing numeric suffix would be attempted until a free username was found. In case of a large number of existing users, this could result in many queries being executed before a free username would be found, potentially resulting in a denial of service. The new algorithm uses a random suffix and only one query to determine the final username.
- Added a new setting: `ACCOUNT_PRESERVE_USERNAME_CASING`. This setting determines whether the username is stored in lowercase (`False`) or whether its casing is to be preserved (`True`). Note that when casing is preserved, potentially expensive `__iexact` lookups are performed when filter on username. For now, the default is set to `True` to maintain backwards compatibility.
- The OAuth2Adapter class has gained a `get_callback_url` method for when customizing the callback URL is desired.
- The Battle.net login backend now accepts the `region` GET parameter.
- New providers: 500px, Discord.

Backwards incompatible changes

- In previous versions, the `DefaultAccountAdapter` contained a `username_regex` property and accompanying `error_messages['invalid_username']` validation error message. These have been removed in favor of using the regex validation already defined at the user model level. Alternatively, you can use the newly introduced `ACCOUNT_USERNAME_VALIDATORS` setting.
- The Battle.net backend no longer overrides username regex validation. In order to use battletags as usernames, you are expected to override either the `username` field on your User model, or to pass a custom validator which will accept the `#` character using the new `ACCOUNT_USERNAME_VALIDATORS` setting. Such a validator is available in `socialaccount.providers.battlenet.validators.BattletagUsernameValidator`.

0.29.0 (2016-11-21)

Note worthy changes

- Addressed Django 1.10 deprecation warnings.

0.28.0 (2016-10-13)

Security notice

- Previous versions contained a vulnerability allowing an attacker to alter the provider specific settings for `SCOPE` and/or `AUTH_PARAMS` (part of the larger `SOCIALACCOUNT_PROVIDERS` setting). The changes would persist across subsequent requests for all users, provided these settings were explicitly set within your project. These settings translate directly into request parameters, giving the attacker undesirable control over the OAuth(2) handshake. You are not affected if you did not explicitly configure these settings. Thanks to Ryan Kelly for reporting!

Note worthy changes

- New providers: Doximity.
- New translations: Korean.

0.27.0 (2016-08-18)

Note worthy changes

- Django 1.10 compatibility.
- The Twitter and GitHub providers now support querying of the email address.

Backwards incompatible changes

- When `ACCOUNT_SIGNUP_EMAIL_ENTER_TWICE` was turned on, the e-mail field key changed from `email` to `email1`, which could introduce subtle bugs. This has now been changed: there always is an `email` field, and optionally an `email2` field.
- The "You must type the same password each time" form validation error that can be triggered during signup is now added to the `password2` field instead of being added to the non field errors.
- The `email_confirmation_sent` signal is now passed `request`, `confirmation` and `signup` instead of only the `confirmation`.
- `ACCOUNT_PASSWORD_MIN_LENGTH` was already deprecated, but is now completely ignored if `AUTH_PASSWORD_VALIDATORS` is not empty.

0.26.1 (2016-07-25)

Note worthy changes

- Locale files wrongly packaged, fixed.
- Fixed bug (`KeyError`) when `ACCOUNT_SIGNUP_EMAIL_ENTER_TWICE` was set to `True`.

0.26.0 (2016-07-24)

Note worthy changes

- New providers: Weixin, Battle.net, Asana, Eve Online, 23andMe, Slack
- Django's password validation mechanism (see `AUTH_PASSWORD_VALIDATORS`) is now used to validate passwords.
- By default, email confirmations are no longer stored in the database. Instead, the email confirmation mail contains an HMAC based key identifying the email address to confirm. The verification lookup includes a fallback to the previous strategy so that there is no negative impact on pending verification emails.
- A new setting `ACCOUNT_SIGNUP_EMAIL_ENTER_TWICE` was added, requiring users to input their email address twice. The setting `ACCOUNT_SIGNUP_PASSWORD_VERIFICATION` has been renamed to `ACCOUNT_SIGNUP_PASSWORD_ENTER_TWICE`.
- New translations: Latvian, Kyrgyz.

Backwards incompatible changes

- Dropped support for Django 1.6
- In order to accomodate for Django's password validation, the `clean_password` method of the adapter now takes an (optional) `user` parameter as its second argument.
- The new HMAC based keys may contain colons. If you have forked `account/urls.py`, be sure to sync the `account_confirm_email` pattern.

0.25.2 (2016-03-13)

Note worthy changes

- Bug fix release (MemcachedKeyCharacterError: "Control characters not allowed")

0.25.1 (2016-03-13)

Note worthy changes

- Bug fix release (AttributeError in password reset view).

0.25.0 (2016-03-12)

Note worthy changes

- Many providers were added: Reddit, Untappd, GitLab, Stripe, Pinterest, Shopify, Draugiem, DigitalOcean, Robinhood, Bitbucket(OAuth2).
- The account connections view is now AJAX aware.
- You can now customize the template extension that is being used to render all HTML templates (`ACCOUNT_TEMPLATE_EXTENSION`)

- In order to be secure by default, users are now blocked from logging in after exceeding a maximum number of failed login attempts (see `ACCOUNT_LOGIN_ATTEMPTS_LIMIT`, `ACCOUNT_LOGIN_ATTEMPTS_TIMEOUT`). Set `ACCOUNT_LOGIN_ATTEMPTS_LIMIT` to `None` to disable this functionality. Important: while this protects the allauth login view, it does not protect Django's admin login from being brute forced.
- New translations: Arabic, Lithuanian

Backwards incompatible changes

None

0.24.1 (2015-11-09)

Note worthy changes

- Non-test code accidentally had test packages as a dependency.

Backwards incompatible changes

- Setting a password after logging in with a social account no longer logs out the user by default on Django 1.7+. Setting an initial password and changing the password both respect `settings.ACCOUNT_LOGOUT_ON_PASSWORD_CHANGE`.

0.24.0 (2015-11-08)

Note worthy changes

- Django 1.9b1 compatibility.
- Seppo Erviälä contributed a Finnish translation, thanks!
- Iurii Kriachko contributed a Basecamp provider, thanks!

Backwards incompatible changes

- Increased `SocialApp` key/secret/token sizes to 191, decreased `SocialAccount.uid` size to 191. The latter was done in order to accomodate for MySQL in combination with utf8mb4 and contraints on `uid`. Note that `uid` is used to store OpenID URLs, which can theoretically be longer than 191 characters, although in practice this does not seem to be the case. In case you really need to control the `uid` length, set `settings.SOCIALACCOUNT_UID_MAX_LENGTH` accordingly. Migrations are in place.

0.23.0 (2015-08-02)

Note worthy changes

- David Friedman contributed Edmodo support, thanks!
- Added support for `ACCOUNT_LOGIN_ON_PASSWORD_RESET` (thanks Julen!)

Backwards incompatible changes

None

0.22.0 (2015-07-23)

Note worthy changes

- Reversal of the email confirmation url can now be overridden in the adapter (`get_email_confirmation_url`). Additionally, the complete confirmation email handling can be overridden via `send_confirmation_mail`.
- Template context processors are no longer used.
- The Facebook Graph API fields (/me/?fields=...) can now be configured using the provider `FIELDS` setting.

Backwards incompatible changes

- Dropped support for Python 2.6 and Django <1.6.
- The default Facebook Graph API version is now v2.4.
- Template context processors are no longer used. The context processor for `allauth.account` was already empty, and the context processor for `allauth.socialaccount` has been converted into the :doc:`{% get_providers %} <templates>` template tag.

0.21.0 (2015-07-02)

Note worthy changes

- You can now tweak the authentication params per OAuth provider, as you already could for OAuth2. Contributed by Peter Rowlands, thanks.
- Nattaphoom Ch. contributed a Thai translation, thanks!
- Guoyu Hao contributed a Baidu provider, thanks!
- Rod Xavier Bondoc contributed support logging out on password change (see setting: `ACCOUNT_LOGOUT_ON_PASSWORD_CHANGE`)

Backwards incompatible changes

- In version 0.20.0 an `account` migration (`0002_email_max_length`) was added to alter the maximum length of the email field. Unfortunately, a side effect of this migration was that the `unique=True` setting slipped through as well. Hardcoding this to `True` is wrong, as uniqueness actually depends on the `ACCOUNT_UNIQUE_EMAIL` setting. We cannot create a followup `0003` migration to set things straight, as the `0002` migration may fail on installations where email addresses are not unique. Therefore, we had to resort to changing an existing migration which is normally not the right thing to do. In case your installation has `ACCOUNT_UNIQUE_EMAIL` set to `True`, you need not take any further action. In case it is set to `False` and migration `0002` already ran, please issue a `--fake` migration down to `0001`, followed by a re-run of the updated `0002`.

0.20.0 (2015-05-25)

Note worthy changes

- Patrick Paul contributed a provider for Evernote, thanks!
- Josh Wright contributed a provider for Spotify, thanks!
- Björn Andersson added support for Dropbox OAuth2, thanks!
- guoqiao contributed a provider for Douban, thanks!

Backwards incompatible changes

- Given that the `max_length` for the Django 1.8 `EmailField` has been bumped to 254, allauth is following up. Migrations (`account`) are in place.

0.19.1 (2015-02-05)

Note worthy changes

- Fixed migrations when using South & Django 1.6.

0.19.0 (2015-01-04)

Note worthy changes

- Basil Shubin contributed an Odnoklassniki provider, thanks!
- Facebook: If the JS SDK is not available, for example due to a browser plugin like Disconnect.me that blocks it, login falls back to the regular non JS handshake.
- `is_safe_url` can now be overriden
- Facebook: The Graph API version is now configurable via `SOCIALACCOUNT_PROVIDERS`.
- A Firefox Accounts provider was added by Jannis Leidel, thanks!
- Josh Owen contributed Coinbase support, thanks!
- Tomas Babej contributed a Slovak translation, thanks!
- Moved existing migrations into `south_migrations`
- "zbryikt" contributed a Taiwanese Chinese translation, thanks!
- Added support for custom password rules via `clean_password`.

Backwards incompatible changes

- In the upcoming Django 1.8 it is no longer possible to hookup an unsaved `User` instance to a `SocialAccount`. Therefore, if you are inspecting the `sociallogin` object, you should now use `sociallogin.user` instead of `sociallogin.account.user`.

- When users logged in while `User.is_active` was `False`, they were sent to `/accounts/inactive/` in case of a social login, and received a form validation error in case of a local login. This needless inconsistency has been removed. The validation error no longer appears and local logins are also redirected to `/accounts/inactive/`.
- In case you were overriding the `ResetPasswordForm`: the save method now takes `request` as its first argument.
- All existing migrations have been moved into `south_migrations` packages, this in order not to conflict with Django's built-in support for migrations. South 1.0 automatically picks up this new location. Upgrade South if you are still dependent on these migrations.

0.18.0 (2014-08-12)

Note worthy changes

- Storing social access tokens is now optional (`SOCIALACCOUNT_STORE_TOKENS`).
- `nimiq` contributed ORCID support, thanks.
- All forms are now pluggable via a new setting: `(SOCIAL)ACCOUNT_FORMS`.
- James Thompson contributed Windows Live support, thanks!

Backwards incompatible changes

- SECURITY: The Persona provider now requires the `AUDIENCE` parameter to be explicitly configured, as required by the Persona specification for security reasons.
- The inline Javascript is removed from the `fbconnect.html` template, which allows for a more strict `Content-Security-Policy`. If you were using the builtin `fbconnect.html` this change should go by unnoticed.

0.17.0 (2014-06-16)

Note worthy changes

- `sourenaraya` contributed Mail.Ru support, thanks.
- account: Justin Michalicek contributed support to control session life time and age: `ACCOUNT_SESSION_COOKIE_AGE` and `ACCOUNT_SESSION_REMEMBER`.
- Serafeim Papastefanos contributed an Ukrainian translation, thanks!
- `kkarwows` contributed AppConfig support, thanks.
- socialaccount: Added Xing provider.
- socialaccount: Marcin Skarbek contributed Hubic support, thanks!
- Volodymyr Yatsyk contributed an Ukrainian translation, thanks!
- `joke2k` contributed an Italian translation, thanks!
- socialaccount: All providers now support the `VERIFIED_EMAIL` property have e-mail addresses forced to be interpreted as verified.

Backwards incompatible changes

None

0.16.1 (2014-03-12)

Note worthy changes

- Facebook login via Javascript was broken if `auth_type` was not set to `reauthenticate`, fixed.
- Support for hooking up a callback when `FB.init()` is ready (`allauth.facebook.onInit`)

Backwards incompatible changes

None

0.16.0 (2014-03-10)

Note worthy changes

- Nariman Gharib contributed a Persian translation, thanks!
- The custom signup form `save` has been deprecated in favour of a `def signup(request, user)` method.
- Facebook reauthentication now uses an `auth_nonce`.
- Added a new option `ACCOUNT_LOGIN_ON_EMAIL_CONFIRMATION`, to indicate whether or not e-mail confirmation is to automatically log in.
- socialaccount: Added Bitbucket provider.
- Jack Shedd contributed Tumblr support, thanks!
- Romanos Tsouroplis contributed Foursquare support, thanks!
- "excessivedemon" contributed Flickr support, thanks!
- Luis Diego García contributed Amazon and Paypal support, thanks!
- Stuart Ross contributed LinkedIn OAuth 2.0 support, thanks!

Backwards incompatible changes

- Previously, the `save(user)` was called on the custom signup form. However, this shadowed the existing `save` method in case a model form was used. To avoid confusion, the `save` method has been deprecated in favour of a `def signup(request, user)` method.
- The Amazon provider requires more space for `token_secret`, so the maximum length restriction has been dropped. Migrations are in place.

0.15.0 (2013-12-01)

Note worthy changes

- socialaccount: Added `is_auto_signup_allowed` to social account adapter.
- facebook: Added a new setting: VERIFIED_EMAIL.
- socialaccount: a collision on e-mail address when you sign up using a third party social account is now more clearly explained: "An account already exists with this e-mail address. Please sign in to that account first, then connect your Google account".
- account: You are now automatically logged in after confirming your e-mail address during sign up.
- account: The `/accounts/login/` view now supports AJAX requests.
- facebook: The fbconnect.js script is now more pluggable.
- socialaccount: Markus Kaiserswerth contributed a Feedly provider, thanks!
- socialaccount: Dropped django-avatar support.
- openid: First, last and full name are now also queried together with the e-mail address. Thanks, @andrvb.
- openid: Compatibility fix for Django 1.6 (JSON serializer).
- account: Added support for `ACCOUNT_CONFIRM_EMAIL_ON_GET`.

Backwards incompatible changes

- Instead of directly rendering and returning a template, logging in while the account is inactive or not yet confirmed now redirects to two new views: `/accounts/inactive/` respectively `/accounts/confirm-email/`.
- The `account/verification_sent.html` template no longer receives the e-mail address in the context (`email`). Note that a message containing that e-mail address is still emitted using the messages framework.
- The `/accounts/confirm_email/key/` view has been renamed to `/accounts/confirm-email/` (human friendlier). Redirects are in place to handle old still pending confirmations.
- Built-in support for django-avatar has been removed. Offering such functionality means making choices which may not be valid for everyone. For example, allauth was downloading the image (which can take some time, or even block) in the context of the login, whereas a better place might be some celery background job. Additionally, in case of an error it simply ignored this. How about retries et al? Also, do you want to copy the avatar once at sign up, or do you want to update on each login? All in all, this functionality goes way beyond authentication and should be addressed elsewhere, beyond allauth scope. The original code has been preserved here so that you can easily reinstate it in your own project: https://gist.github.com/pennersr/7571752

0.14.2 (2013-11-16)

Note worthy changes

- Compatibility fix for logging in with Django 1.6.
- Maksim Rukomoynikov contributed a Russian translation, thanks!

Backwards incompatible changes

- In case you were using the internal method `generate_unique_username`, note that its signature has changed. It now takes a list of candidates to base the username on.

0.14.1 (2013-10-28)

Note worthy changes

- PyPi did not render the README.rst properly.

Backwards incompatible changes

None

0.14.0 (2013-10-28)

Note worthy changes

- Stuart Ross contributed AngelList support, thanks!
- LinkedIn: profile fields that are to be fetched are now configurable (`PROFILE_FIELDS` provider-level setting).
- Udi Oron contributed a Hebrew translation, thanks!
- Add setting `ACCOUNT_DEFAULT_HTTP_PROTOCOL` (HTTPS support).
- George Whewell contributed Instagram support, thanks!
- Refactored adapter methods relating to creating and populating `User` instances.
- User creation methods in the `Default(Social)AccountAdapter` now have access to the `request`.

Backwards incompatible changes

- The `socialaccount/account_inactive.html` template has been moved to `account/account_inactive.html`.
- The adapter API for creating and populating users has been overhauled. As a result, the `populate_new_user` adapter methods have disappeared. Please refer to the section on "Creating and Populating User Instances" for more information.

0.13.0 (2013-08-31)

Note worthy changes

- Koichi Harakawa contributed a Japanese translation, thanks!
- Added `is_open_for_signup` to DefaultSocialAccountAdapter.
- Added VK provider support.
- Marcin Spoczynski contributed a Polish translation, thanks!
- All views are now class-based.

- `django.contrib.messages` is now optional.
- "jresins" contributed a simplified Chinese, thanks!

Backwards incompatible changes

- The password reset from key success response now redirects to a "done" view (`/accounts/password/reset/key/done/`). This view has its own `account/password_reset_from_key_done.html` template. In previous versions, the success template was intertwined with the `account/password_reset_from_key.html` template.

0.12.0 (2013-07-01)

Note worthy changes

- Added support for re-authenticated (forced prompt) by means of a new `action="reauthenticate"` parameter to the `{% provider_login_url %}`
- Roberto Novaes contributed a Brazilian Portuguese translation, thanks!
- Daniel Eriksson contributed a Swedish translation, thanks!
- You can now logout from both allauth and Facebook via a Javascript helper: `window.allauth.facebook.logout()`.
- Connecting a social account is now a flow that needs to be explicitly triggered, by means of a `process="connect"` parameter that can be passed along to the `{% provider_login_url %}`, or a `process=connect` GET parameter.
- Tomas Marcik contributed a Czech translation, thanks!

Backwards incompatible changes

- The `{% provider_login_url %}` tag now takes an optional process parameter that indicates how to process the social login. As a result, if you include the template `socialaccount/snippets/provider_list.html` from your own overriden `socialaccount/connections.html` template, you now need to pass along the process parameter as follows: `{% include "socialaccount/snippets/provider_list.html" with process="connect" %}`.
- Instead of inlining the required Facebook SDK Javascript wrapper code into the HTML, it now resides into its own .js file (served with `{% static %}`). If you were using the builtin `fbconnect.html` this change should go by unnoticed.

0.11.1 (2013-06-04)

Note worthy changes

- Released (due to issue in disconnecting social accounts).

Backwards incompatible changes

None

0.11.0 (2013-06-02)

Note worthy changes

- Moved logic whether or not a social account can be disconnected to the `SocialAccountAdapter` (`validate_disconnect`).
- Added `social_account_removed` signal.
- Implemented CSRF protection (http://tools.ietf.org/html/draft-ietf-oauth-v2-30#section-10.12).
- The `user_logged_in` signal now optionally receives a `sociallogin` parameter, in case of a social login.
- Added `social_account_added` (contributed by orblivion, thanks).
- Hatem Nassrat contributed Bitly support, thanks!
- Bojan Mihelac contributed a Croatian translation, thanks!
- Messages (as in `django.contrib.messages`) are now configurable through templates.
- Added support for differentiating e-mail handling (verification, required) between local and social accounts: `SOCIALACCOUNT_EMAIL_REQUIRED` and `SOCIALACCOUNT_EMAIL_VERIFICATION`.

Backwards incompatible changes

None

0.10.1 (2013-04-16)

Note worthy changes

- Cleaning of `username` can now be overriden via `DefaultAccountAdapter.clean_username`
- Fixed potential error (`assert`) when connecting social accounts.
- Added support for custom username handling in case of custom user models (`ACCOUNT_USER_MODEL_USERNAME_FIELD`).

Backwards incompatible changes

None

0.10.0 (2013-04-12)

Note worthy changes

- Chris Davis contributed Vimeo support, thanks!
- Added support for overriding the URL to return to after connecting a social account (`allauth.socialaccount.adapter.DefaultSocialAccountAdapter.get_connect_redirect_url`).
- Python 3 is now supported!
- Dropped dependency on (unmaintained?) oauth2 package, in favor of requests-oauthlib.

- account: E-mail confirmation mails generated at signup can now be differentiated from regular e-mail confirmation mails by placing e.g. a welcome message into the `account/email/email_confirmation_signup*` templates. Thanks to Sam Solomon for the patch.
- account: Moved User instance creation to adapter so that e.g. username generation can be influenced. Thanks to John Bazik for the patch.
- Robert Balfre contributed Dropbox support, thanks!
- socialaccount: Added support for Weibo.
- account: Added support for sending HTML e-mail. Add `*_message.html` templates and they will be automatically picked up.
- Added support for passing along extra parameters to the OAuth2 authentication calls, such as `access_type` (Google) or `auth_type` (Facebook).
- Both the login and signup view now immediately redirect to the login redirect url in case the user was already authenticated.
- Added support for closing down signups in a pluggable fashion, making it easy to hookup your own invitation handling mechanism.
- Added support for passing along extra parameters to the `FB.login` API call.

Backwards incompatible changes

- Logout no longer happens on GET request. Refer to the `LogoutView` documentation for more background information. Logging out on GET can be restored by the setting `ACCOUNT_LOGOUT_ON_GET`. Furthermore, after logging out you are now redirected to `ACCOUNT_LOGOUT_REDIRECT_URL` instead of rendering the `account/logout.html` template.
- `LOGIN_REDIRECT_URLNAME` is now deprecated. Django 1.5 accepts both URL names and URLs for `LOGIN_REDIRECT_URL`, so we do so as well.
- `DefaultAccountAdapter.stash_email_verified` is now named `stash_verified_email`.
- Django 1.4.3 is now the minimal requirement.
- Dropped dependency on (unmaintained?) oauth2 package, in favor of requests-oauthlib. So you will need to update your (virtual) environment accordingly.
- We noticed a very rare bug that affects end users who add Google social login to existing accounts. The symptom is you end up with users who have multiple primary email addresses which conflicts with assumptions made by the code. In addition to fixing the code that allowed duplicates to occur, there is a managegement command you can run if you think this effects you (and if it doesn't effect you there is no harm in running it anyways if you are unsure):
  - `python manage.py account_unsetmultipleprimaryemails`
    - Will silently remove primary flags for email addresses that aren't the same as `user.email`.
    - If no primary `EmailAddress` is `user.email` it will pick one at random and print a warning.
- The expiry time, if any, is now stored in a new column `SocialToken.expires_at`. Migrations are in place.
- Furthermore, Facebook started returning longer tokens, so the maximum token length was increased. Again, migrations are in place.
- Login and signup views have been turned into class-based views.

- The template variable `facebook_perms` is no longer passed to the "facebook/fbconnect.html" template. Instead, `fb_login_options` containing all options is passed.

0.9.0 (2013-01-30)

Note worthy changes

- account: `user_signed_up` signal now emits an optional `sociallogin` parameter so that receivers can easily differentiate between local and social signups.
- account: Added `email_removed` signal.
- socialaccount: Populating of User model fields is now centralized in the adapter, splitting up `name` into `first_name` and `last_name` if these were not individually available.
- Ahmet Emre Aladağ contributed a Turkish translation, thanks!
- socialaccount: Added SocialAccountAdapter hook to allow for intervention in social logins.
- google: support for Google's `verified_email` flag to determine whether or not to send confirmation e-mails.
- Fábio Santos contributed a Portugese translation, thanks!
- socialaccount: Added support for Stack Exchange.
- socialaccount: Added `get_social_accounts` template tag.
- account: Default URL to redirect to after login can now be overriden via the adapter, both for login and e-mail confirmation redirects.

Backwards incompatible changes

- `requests` is now a dependency (dropped `httplib2`).
- Added a new column `SocialApp.client_id`. The value of `key` needs to be moved to the new `client_id` column. The `key` column is required for Stack Exchange. Migrations are in place to handle all of this automatically.

0.8.3 (2012-12-06)

Note worthy changes

- Markus Thielen contributed a German translation, thanks!
- The `site` foreign key from `SocialApp` to `Site` has been replaced by a `ManyToManyField`. Many apps can be used across multiple domains (Facebook cannot).
- account: Added adapter class for increased pluggability. Added hook for 3rd party invitation system to by pass e-mail verification (`stash_email_verified`). Moved sending of mail to adapter.
- account: Added option to completely disable e-mail verification during signup.

Backwards incompatible changes

- The `ACCOUNT_EMAIL_VERIFICATION` setting is no longer a boolean based setting. Use a string value of "none", "optional" or "mandatory" instead.
- The template "account/password_reset_key_message.txt" has been moved to "account/email/password_reset_key_message.txt". The subject of the message has been moved into a template ("account/email/password_reset_key_subject.txt").
- The `site` foreign key from `SocialApp` to `Site` has been replaced by a `ManyToManyField`. Many apps can be used across multiple domains (Facebook cannot).

0.8.2 (2012-10-10)

Note worthy changes

- Twitter: Login was broken due to change at in URLs at Twitter, fixed.
- LinkedIn: Added support for passing along the OAuth scope.
- account: Improved e-mail confirmation error handling, no more confusing 404s.
- account: Aldiantoro Nugroho contributed support for a new setting: ACCOUNT_USERNAME_MIN_LENGTH
- socialaccount: Added preliminary support for Mozilla Persona.
- account: Sam Solomon added various signals for email and password related changes.
- account: Usernames may now contain @, +, . and - characters.

Backwards incompatible changes

- Dropped support for `CONTACT_EMAIL` from the `account` template context processor. It was never documented and only used in the templates as an example – there is no need to pollute the `allauth` settings with that. If your templates rely on it then you will have to put it in a context processor yourself.

0.8.1 (2012-09-03)

Note worthy changes

- Python 2.6.2 compatibility issue, fixed.
- The example project was unintentionally packaged, fixed.

Backwards incompatible changes

None

0.8.0 (2012-09-01)

Note worthy changes

- account: Dropped dependency on the emailconfirmation app, integrating its functionality into the account app. This change is of major impact, please refer to the documentation on how to upgrade.
- account: Documented ACCOUNT_USERNAME_REQUIRED. This is actually not a new setting, but it somehow got overlooked in the documentation.
- account/socialaccount: Dropped the _tags postfix from the template tag libraries. Simply use {% load account %} and {% load socialaccount %}.
- Added signup and social login signals.
- SoundCloud: Rabi Alam contributed a SoundCloud provider, thanks!
- account: Sam Solomon cleaned up the e-mail management view: added proper redirect after POSTs, prevent deletion of primary e-mail. Thanks.
- account: When signing up, instead of generating a completely random username a more sensible username is automatically derived from first/last name or e-mail.

Backwards incompatible changes

- `allauth` now depends on Django 1.4 or higher.
- Major impact: dropped dependency on the `emailconfirmation` app, as this project is clearly left unmaintained. Important tickets such as https://github.com/pinax/django-email-confirmation/pull/5 are not being addressed. All models and related functionality have been directly integrated into the `allauth.account` app. When upgrading take care of the following:
  - The `emailconfirmation` setting `EMAIL_CONFIRMATION_DAYS` has been replaced by `ACCOUNT_EMAIL_CONFIRMATION_EXPIRE_DAYS`.
  - Instead of directly confirming the e-mail address upon the GET request the confirmation is now processed as part of an explicit POST. Therefore, a new template `account/email_confirm.html` must be setup.
  - Existing `emailconfirmation` data should be migrated to the new tables. For this purpose a special management command is available: `python manage.py account_emailconfirmationmigration`. This command does not drop the old `emailconfirmation` tables – you will have to do this manually yourself. Why not use South? EmailAddress uniqueness depends on the configuration (`ACCOUNT_UNIQUE_EMAIL`), South does not handle settings dependent database models.
- `{% load account_tags %}` is deprecated, simply use: `{% load account %}`
- `{% load socialaccount_tags %}` is deprecated, simply use: `{% load socialaccount %}`

0.7.0 (2012-07-18)

Note worthy changes

- Facebook: Facundo Gaich contributed support for dynamically deriving the Facebook locale from the Django locale, thanks!.

- OAuth: All OAuth/OAuth2 tokens are now consistently stored across the board. Cleaned up OAuth flow removing superfluous redirect.
- Facebook: Dropped Facebook SDK dependency.
- socialaccount: DRY focused refactoring of social login.
- socialaccount: Added support for Google OAuth2 and Facebook OAuth2. Fixed GitHub.
- account: Added verified_email_required decorator.
- socialaccount: When signing up, user.first/last_name where always taken from the provider signup data, even when a custom signup form was in place that offered user inputs for editting these fields. Fixed.

Backwards incompatible changes

None

0.6.0 (2012-06-20)

Note worthy changes

- account: Added ACCOUNT_USER_DISPLAY to render a user name without making assumptions on how the user is represented.
- allauth, socialaccount: Removed the last remaining bits of hardcodedness with respect to the enabled social authentication providers.
- account: Added ACCOUNT_AUTHENTICATION_METHOD setting, supporting login by username, e-mail or both.

Backwards incompatible changes

- The `ACCOUNT_EMAIL_AUTHENTICATION` setting has been dropped in favor of `ACCOUNT_AUTHENTICATION_METHOD`.
- The login form field is now always named `login`. This used to by either `username` or `email`, depending on the authentication method. If needed, update your templates accordingly.
- The `allauth` template tags (containing template tags for OpenID, Twitter and Facebook) have been removed. Use the `socialaccount` template tags instead (specifically: `{% provider_login_url ... %}`).
- The `allauth.context_processors.allauth` context processor has been removed, in favor of `allauth.socialaccount.context_processors.socialaccount`. In doing so, all hardcodedness with respect to providers (e.g `allauth.facebook_enabled`) has been removed.

0.5.0 (2012-06-08)

Note worthy changes

- account: Added setting ACCOUNT_PASSWORD_MIN_LENGTH for specifying the minimum password length.
- socialaccount: Added generic OAuth2 support. Added GitHub support as proof of concept.
- socialaccount: More refactoring: generic provider & OAuth consumer approach. Added LinkedIn support to test this approach.

- socialaccount: Introduced generic models for storing social apps, accounts and tokens in a central and consistent manner, making way for adding support for more account providers. Note: there is more refactoring to be done – this first step only focuses on the database models.
- account: E-mail confirmation mails are now automatically resent whenever a user attempts to login with an unverified e-mail address (if ACCOUNT_EMAIL_VERIFICATION=True).

Backwards incompatible changes

- Upgrade your `settings.INSTALLED_APPS`: Replace `allauth.<provider>` (where provider is one of `twitter`, `facebook` or `openid`) with `allauth.socialaccount.providers.<provider>`
- All provider related models (`FacebookAccount`, `FacebookApp`, `TwitterAccount`, `TwitterApp`, `OpenIDAccount`) have been unified into generic `SocialApp` and `SocialAccount` models. South migrations are in place to move the data over to the new models, after which the original tables are dropped. Therefore, be sure to run migrate using South.

0.4.0 (2012-03-25)

Note worthy changes

- account: The render_value parameter of all PasswordInput fields used can now be configured via a setting.
- account: Added support for prefixing the subject of sent emails.
- account: Added support for a plugging in a custom signup form used for additional questions to ask during signup.
- account: `is_active` is no longer used to keep users with an unverified e-mail address from loging in.
- Dropping uniform dependency. Moved uniform templates into example project.

Backwards incompatible changes

None

0.3.0 (2012-01-19)

Note worthy changes

- The e-mail authentication backend now attempts to use the 'username' parameter as an e-mail address. This is needed to properly integrate with other apps invoking authenticate.
- SmileyChris contributed support for automatically generating a user name at signup when `ACCOUNT_USERNAME_REQUIRED` is set to False.
- Vuong Nguyen contributed support for (optionally) asking for the password just once during signup (`ACCOUNT_SIGNUP_PASSWORD_VERIFICATION`).
- The Twitter oauth sequence now respects the "oauth_callback" parameter instead of defaulting to the callback URL configured at Twitter.
- Pass along `?next=` parameter between login and signup views.
- Added Dutch translation.

- Added template tags for pointing to social login URLs. These tags automatically pass along any `?next=` parameter. Additionally, added an overall allauth_tags that gracefully degrades when e.g. allauth.facebook is not installed.
- Pass along next URL, if any, at `/accounts/social/signup/`.
- Duplicate email address handling could throw a MultipleObjectsReturned exception, fixed.
- Removed separate social account login view, in favour of having a single unified login view including both forms of login.
- Added support for passing along a next URL parameter to Facebook, OpenID logins.
- Added support for django-avatar, copying the Twitter profile image locally on signup.
- `allauth/account/forms.py` (`BaseSignupForm.clean_email`): With `ACCOUNT_EMAIL_REQUIRED=False`, empty email addresses were considered duplicates. Fixed.
- The existing migrations for allauth.openid were not compatible with MySQL due to the use of an URLField with max_length above 255. The issue has now been addressed but unfortunately at the cost of the existing migrations for this app. Existing installations will have to be dealt with manually (altering the "identity" column of OpenIDAccount, deleting ghost migrations).

Backwards incompatible changes

- None

Commercial Support

This project is sponsored by IntenCT. If you require assistance on your project(s), please contact us: info@intenct.nl.

Cross-Selling

If you like this, you may also like:

- django-trackstats: https://github.com/pennersr/django-trackstats
- netwell: https://github.com/pennersr/netwell

CHAPTER 5

Indices and tables

- genindex
- search