django-allauth-cas Documentation

Release 1.0.0b2

Aurélien Delobelle

Dec 29, 2017

Contents

CAS support for django-allauth.

Requirements

- Django 1.8 → 2.0

Dependencies

- django-allauth
- python-cas: CAS client library

Note: Tests only target the latest allauth version compatible for each Django version supported:

- Django 1.9 with django-allauth 0.32.0;
- Django 1.8, 1.10, 1.11, 2.0 with the latest django-allauth.

If you have any problems at use or think docs can be clearer, take a little time to open an issue and/or a PR would be welcomed ;-)

Acknowledgments

- This work is strongly inspired by the OAuth2 support of django-allauth.

CHAPTER 1

Installation

Install the python package `django-allauth-cas`. For example, using pip:

```
$ pip install django-allauth-cas
```

Add `'allauth_cas'` to `INSTALLED_APPS`.

CHAPTER 2

Contents

2.1 Basic setup

Following the instructions on this page, your will create a provider for allauth, which allows users to connect through a CAS server.

2.1.1 1. Create an app

`allauth` determines available providers by scanning `INSTALLED_APPS`. Let's begin by creating an app for the CAS provider:

```
$ python manage.py startapp mycas
```

And add it to the `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    # ...
    'allauth',
    'allauth.account',
    'allauth.socialaccount',

    'allauth_cas',

    'mycas',
]
```

2.1.2 2. Create the provider

In `mycas/provider.py`, create subclasses of `ProviderAccount` and `CASProvider`.

The `CASProvider` subclass defines how to process data returned by the CAS server.

```
from allauth.socialaccount.providers.base import ProviderAccount
from allauth_cas.providers import CASProvider


class MyCASAccount(ProviderAccount):
    pass


class MyCASProvider(CASProvider):
    id = 'mycas'  # Choose an identifier for your provider
    name = 'My CAS'  # Verbose name of your provider
    account_class = MyCASAccount


provider_classes = [ClipperProvider]
```

See also:

Use data returned by a CAS server

2.1.3 3. Create the views

Subclass `CASAdapter` to give your configuration as a CAS client.

```
from allauth_cas.views import CASAdapter

from .providers import MyCASProvider


class MyCASAdapter(CASAdapter):
    provider_id = MyCASProvider.id
    url = 'https://mycas.mydomain.net'  # The CAS server url
    version = 3  # Select the CAS protocol version used by the CAS server: 1, 2, 3...
```

Then, you can simply create the login and callback views.

```
from allauth_cas.views import CASCallbackView, CASLoginView

login = CASLoginView.adapter_view(MyCASAdapter)
callback = CASLogoutView.adapter_view(MyCASAdapter)
```

See also:

Configure the CAS client

2.1.4 4. Create the urls

Finally, add the urls in `mycas/urls.py`.

```
from allauth_cas.urls import default_urlpatterns

from .provider import MyCASProvider

urlpatterns = default_urlpatterns(MyCasProvider)
```

There is no need to do more, as `allauth` is responsible for including these urls.

2.1.5 5. Allow your application at the CAS server

Note: This step is only required if the CAS server restricts access to known applications.

CAS servers may restrict their usage to a list of known clients. To do so, the service url must be known by the CAS server. For our case, the service url is the callback url of a CAS provider.

The service url is formatted as:

```
<url of your application>/<path to allauth urls>/<provider id>/login/callback/
```

Assuming a site is served at `https://mydomain.net`, that the allauth urls are included under `accounts/`, and the provider id is `mycas`, the service url is:

```
https://mydomain.net/accounts/mycas/login/callback
```

While in local development, it can be:

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

This url should be added to the authorized services within the CAS server configuration (by yourself or someone in charge of the server).

2.2 Advanced Usage

2.2.1 Configure the CAS client

See also:

CAS Protocol Specification

The CAS client parameters can be set on the `CASAdapter` subclasses.

Server information

You must at least fill these attributes on an adapter class.

`CASAdapter.`**`url = None`**

`CASAdapter.`**`version = None`**

Client parameters

`CASAdapter.`**`renew`**

Controls presence of `renew` in requests to the CAS server.

If `True`, opt out single sign-on (SSO) functionality of the CAS server. So that, user is always prompted for his username and password.

If `False`, the CAS server does not prompt users for their credentials if a SSO exists.

The default allows user to connect via an already used CAS server with other credentials.

Returns `True` if logged in user has already connected to Django using **any** CAS provider in the current session, `False` otherwise.

Note: A SSO session is created when user successfully authenticates against the server, which let an HTTP cookie in the browser current session. If SSO is enabled (`renew = False`), server checks this cookie, if any, to bypass the request of user credentials. Depending on the server configuration and user input at login time, CAS server replies to login page requests with a warning page, or transparently redirects to the callback url (path to come back to your web service).

2.2.2 Use data returned by a CAS server

See also:

Creating and Populating User instances

The following methods of `CASProvider` are used to extract data from the CAS responses.

`CASProvider.`**`extract_uid`**`(`*data*`)`
Extract the user uid.

Notes

Each pair `(provider_id, uid)` is unique and related to a single user.

Parameters **`data`** (*`uid (str), extra (dict)`*) – CAS response. Example: `('alice', {'name': 'Alice'})`

Returns Default to `data[0]`, user identifier for the CAS server.

Return type str

`CASProvider.`**`extract_common_fields`**`(`*data*`)`
Extract the data to pass to *SOCIALACCOUNT_ADAPTER.populate_user()*.

Parameters **`data`** (*`uid (str), extra (dict)`*) – CAS response. Example: `('alice', {'name': 'Alice'})`

Returns

Default:

```
{
    'username': extra.get('username', uid),
    'email': extra.get('email'),
    'first_name': extra.get('first_name'),
    'last_name': extra.get('last_name'),
    'name': extra.get('name'),
}
```

Return type dict

`CASProvider.`**`extract_email_addresses`**`(`*data*`)`
Extract the email addresses.

Parameters **`data`** (*`uid (str), extra (dict)`*) – CAS response. Example: `('alice', {'name': 'Alice'})`

Returns

By default, `[]`.

Example:

```
[
    EmailAddress(
        email='user@domain.net',
        verified=True, primary=True,
    ),
    EmailAddress(
        email='alias@domain.net',
        verified=True, primary=False,
    ),
]
```

Return type *list* of *EmailAddress*

`CASProvider.`**`extract_extra_data`**`(`*data*`)`

Extract the data to save to *SocialAccount.extra_data*.

Parameters **`data`** (*`uid (str), extra (dict)`*) – CAS response. Example: `('alice', {'name': 'Alice'})`

Returns By default, `data`.

Return type dict

2.2.3 Sign out helpers

To use features described on this page, you must also add a logout view for your provider:

```
from allauth_cas.views import CASLogoutView

logout = CASLogoutView.adapter_view(MyCASAdapter)
```

Suggest logout

Sending message

Using the method below, you can emit a message to suggest users to logout of the CAS server.

`CASProvider.`**`add_message_suggest_caslogout`**`(`*request*, *next_page=None*, *level=None*`)`

Add a message with a link for the user to logout of the CAS server.

It uses the template `socialaccount/messages/suggest_caslogout.html`, with the `provider` and the `logout_url` as context.

Parameters

- **`request`** – The request to which the message is added.
- **`next_page`** (*`optional`*) – Added to the logout link for the CAS server to redirect the user to this url. Default: `request.get_full_path()`
- **`level`** – The message level. Default: `messages.INFO`

Sending message at user logout

When the user signs out your application, this message can be sent automatically using the following settings.

The message contains a logout link for **the last used** CAS server during the session.

In your settings:

```
SOCIALACCOUNT_PROVIDERS = {
    # ...
    '<provider id>': {
        # ...

        'MESSAGE_SUGGEST_CASLOGOUT_ON_LOGOUT': True,

        # Optional. By default, messages.INFO
        'MESSAGE_SUGGEST_CASLOGOUT_ON_LOGOUT_LEVEL': messages.WARNING,
    },
}
```

If you need more control over the sending of the message, you can use the methods below of the provider class.

`CASProvider.`**`message_suggest_caslogout_on_logout`**(*request*)
Indicates whether the logout message should be sent on user logout.

By default, it returns `settings.SOCIALACCOUNT_PROVIDERS[self.id]['MESSAGE_SUGGEST_CASLOGOUT_ON_LOGOUT']` or `False`.

Notes

The `request` argument is the one trigerring the emission of the signal `user_logged_out`.

`CASProvider.`**`message_suggest_caslogout_on_logout_level`**(*request*)
Level of the logout message issued on user logout.

By default, it returns `settings.SOCIALACCOUNT_PROVIDERS[self.id]['MESSAGE_SUGGEST_CASLOGOUT_ON_LOGOUT_LEVEL']` or `messages.INFO`.

Notes

The `request` argument is the one trigerring the emission of the signal `user_logged_out`.

Redirection after CAS logout

An url is always given for the CAS server to redirect the user to your application.

The target of this redirection is:

- If the link is created on user logout (using above configuration):
  - if present, the url pointed by the GET parameter `next`, which should be the url the user has just landed after being logged out;
  - otherwise, the value returned by `ACCOUNT_ADAPTER.get_logout_redirect_url()`.
- If the link is created using *`add_message_suggest_caslogout()`*:
  - if present, the value of the parameter `next_page`;

  - otherwise, the url of the current page.
- Otherwise, `ACCOUNT_ADAPTER.get_logout_redirect_url()`.

Note: If no redirection happens, you should check the version declared by the `CASAdapter` class corresponds to the CAS server one.

2.3 Changelog

2.3.1 1.0.0 (unreleased)

- First official release.

Index