Django Keycloak Documentation

Release 0.1.1

Peter Slump

Feb 26, 2019

Scenario's

CHAPTER 1

Testing with the Example project

The quickest way to experiment with this project is by running the example project. This project is setup using Docker compose.

Given you have installed Docker and Docker compose run:

```
$ cd example
$ docker-compose up
```

The project exists of a resource provider which mimics a web app and a resource provider which is only accessible by an API. Next to it is a Keycloak instance available which is backed by a Postgres database.

Once you have the containers running you can access it by navigating to: https://resource-provider.localhost.yarf.nl/ you can login with username: *testuser* and password: *password*. The admin is accessible at /admin with username: *admin* and password: *password*.

The Keycloak instance is available at: https://identity.localhost.yarf.nl/ the username of the admin user is *admin* and the password is *password*.

The API is available at: https://resource-provider-api.localhost.yarf.nl/ You probably don't actually use this server or only for the admin. The admin is accessible at /admin with username: *admin* and password: *password*.

CHAPTER 2

Setup for local user storage

By local user storage a User object get created for every logged in identity. This can be handy when you want to link objects to this User. If that's not the case please read the scenario *Setup for remote user*.

Since **this is the default behaviour for Django Keycloak** you don't have to configure any setting.

Important to point out the *KEYCLOAK_OIDC_PROFILE_MODEL* setting. This should contain *django_keycloak.OpenIdConnectProfile* (which is the case by default). The model to store the Open ID Connect profile is a swappable model. When configured to this model a foreign key to the configured Django User model is available.

```
# settings.py
KEYCLOAK_OIDC_PROFILE_MODEL = 'django_keycloak.OpenIdConnectProfile'
```

CHAPTER 3

Setup for remote user

It's not required to create a local User object for every logged in identity. If you don't need a local user object you can setup the app to work with a remote user. This user behaves like Django's User object but it is not a real one.

Note: For logging purposes Django admin only works with User objects which are stored in the database. So you cannot use this method to authenticate users for admin usage.

Warning: Set the configuration setting below before running the migrations!

Set the OIDC Profile model to the remote variant:

```
# your-project/settings.py
KEYCLOAK_OIDC_PROFILE_MODEL = 'django_keycloak.RemoteUserOpenIdConnectProfile'
```

Configure the remote user middleware:

```
MIDDLEWARE = [
    ...

    'django_keycloak.middleware.BaseKeycloakMiddleware',
    'django_keycloak.middleware.RemoteUserAuthenticationMiddleware',
]
```

By default the class *django_keycloak.remote_user.KeycloakRemoteUser* is used as user, this one will be available on the request when authenticated and will be returned when you access *RemoteUserOpenIdConnectProfile.user*. If you want another class (i.e. you need extra properties) you can configure this class using the setting *KEYCLOAK_REMOTE_USER_MODEL*:

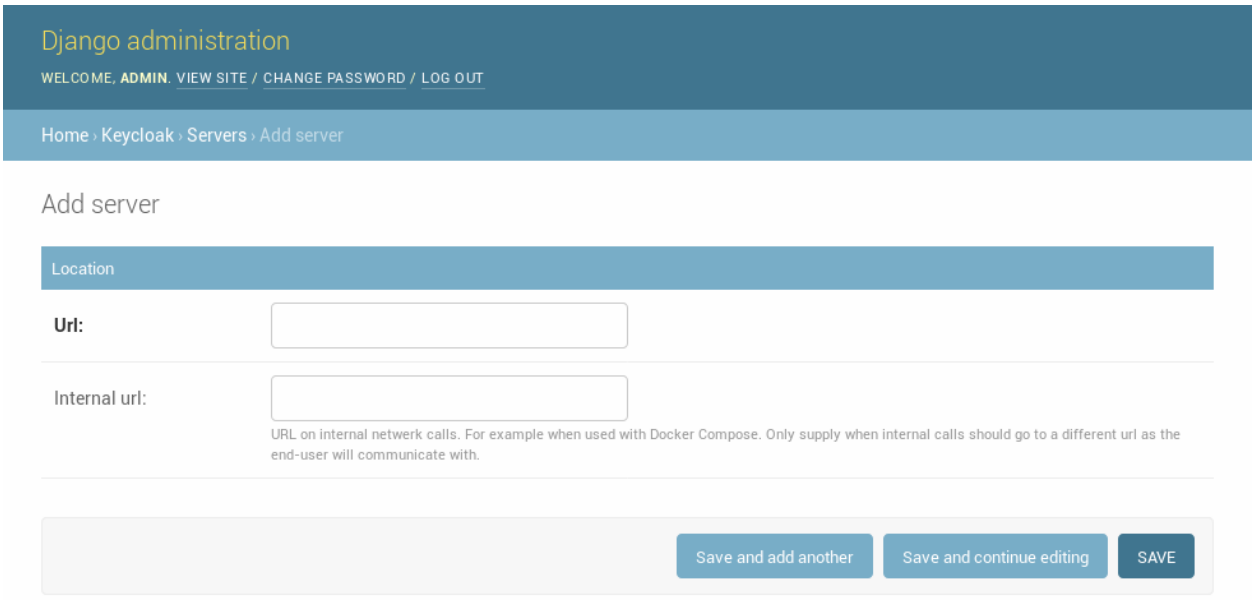
```
KEYCLOAK_REMOTE_USER_MODEL = 'django_keycloak.remote_user.KeycloakRemoteUser'
```

CHAPTER 4

Initial setup

4.1 Server configuration

First you have to add your Keycloak server. You can do this in the Django Admin.

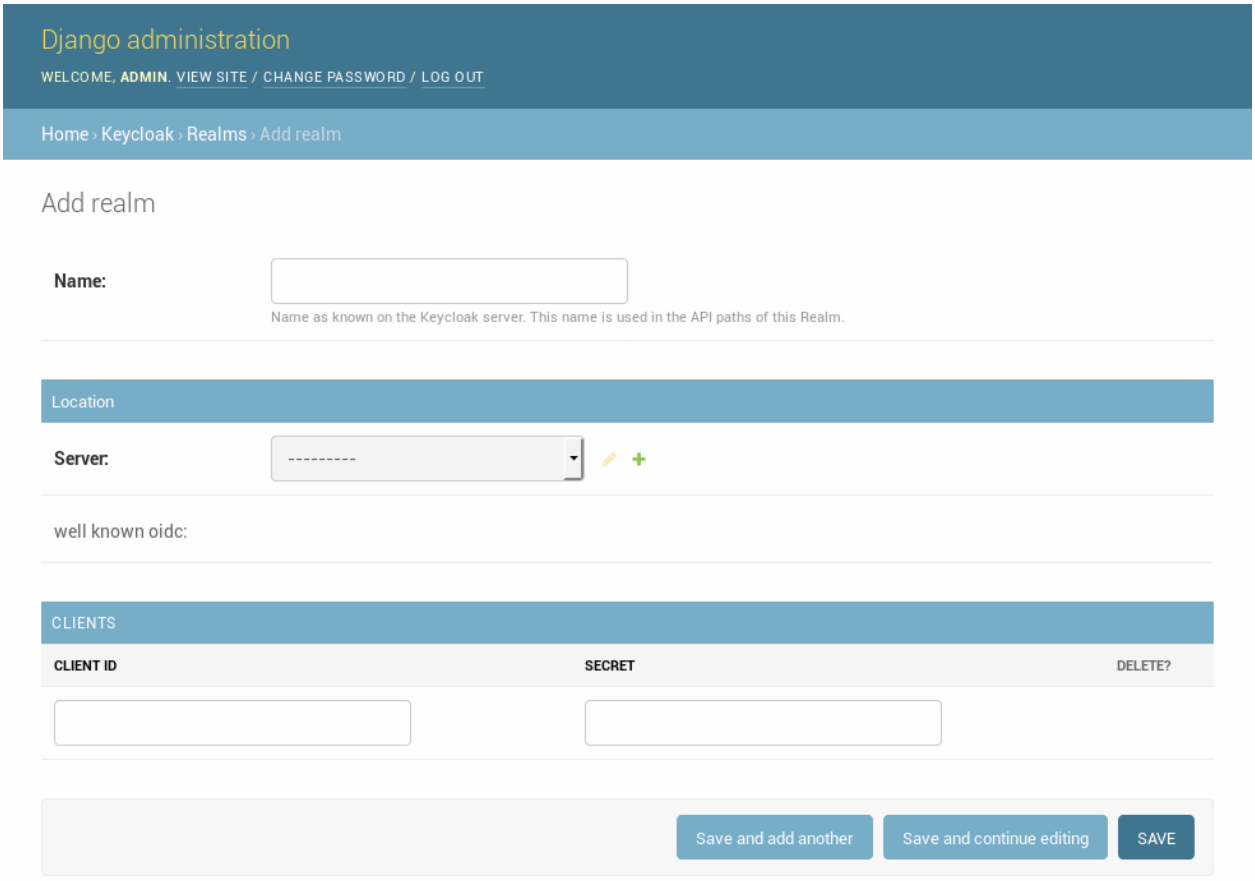


Note: When your application access the Keycloak server using a different url than the public one you can configure this URL as "internal url". Django Keycloak will use that url for all direct communication but uses the standard server url to redirect users for authentication.

4.2 Realm configuration

After you have created a REALM and Client in Keycloak you can add these in the Django admin.

Note: Django-Keycloak supports multiple realms. However when you configure multiple realms you have to write your own middleware which selects the correct realm based on the request. The default middleware always selects the first realm available in the database.



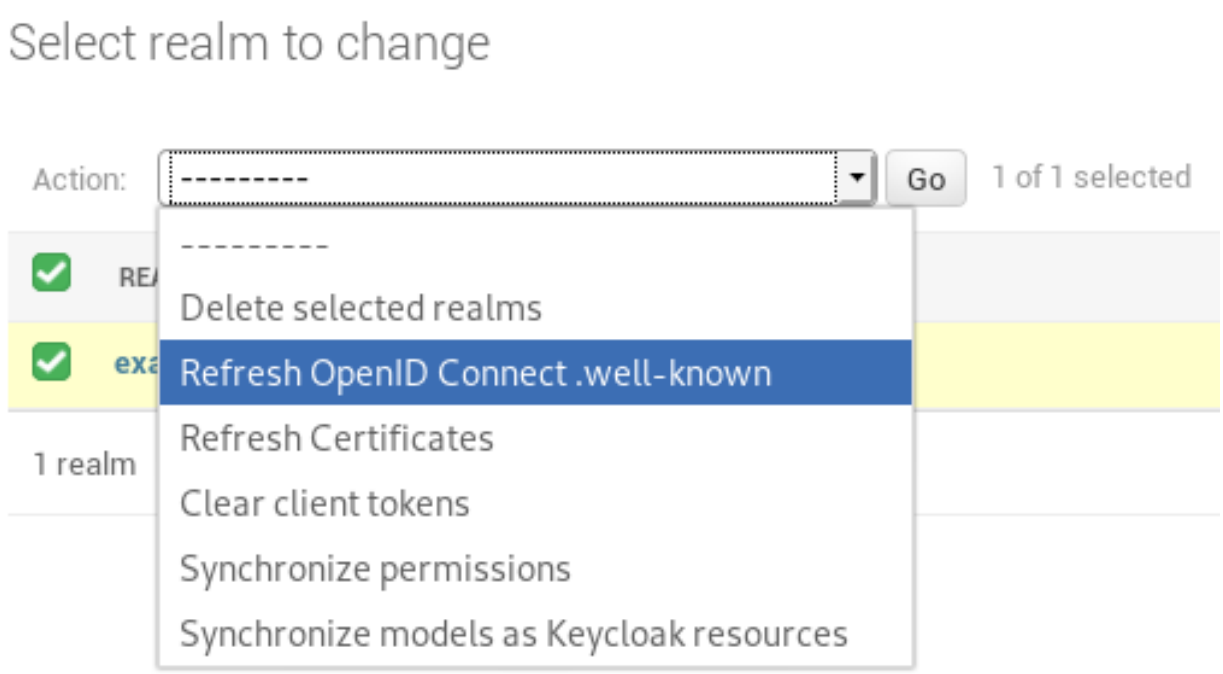
After you have added the realm please make sure to run te following actions:

- *Refresh OpenID Connect .well-known*
- *Refresh Certificates*
- synchronize_permissions (when using the permission system)

4.3 Tools

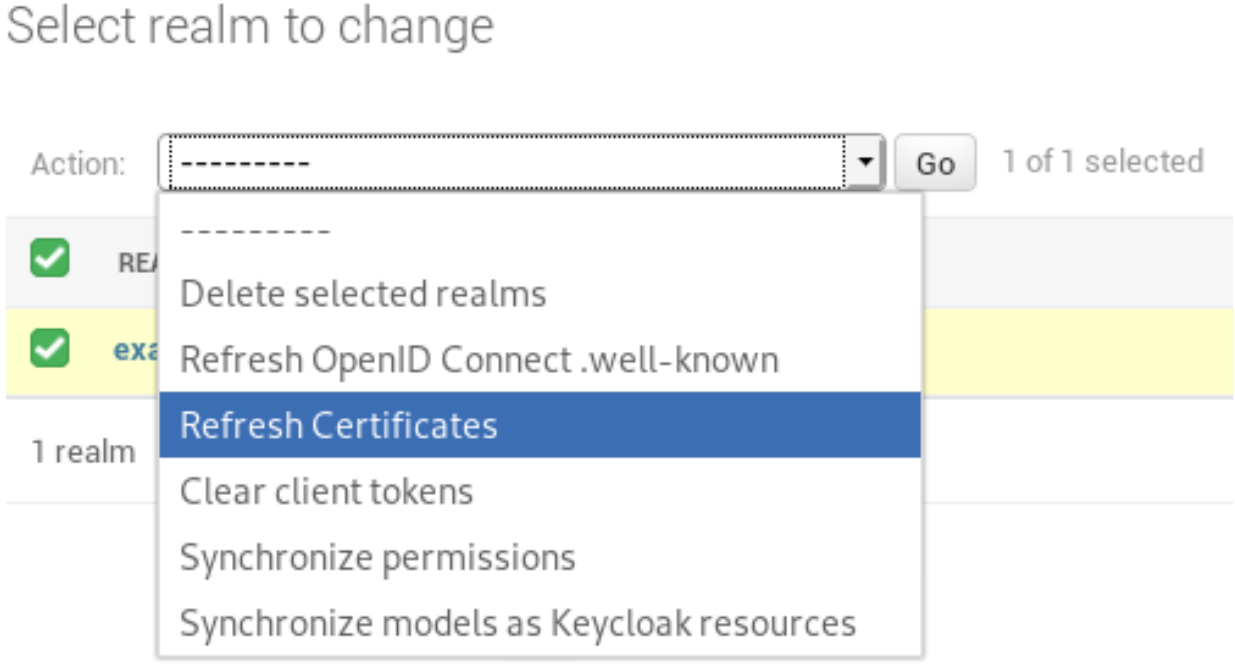
4.3.1 Refresh OpenID Connect .well-known

In the Django Admin you can apply the action "Refresh OpenID Connect .well-known" for a realm. This retrieves the .well-known content for the OpenID Connect functionality and caches this in the database. In this way it's not required to fetch this file before each request regarding OpenID Connect to the Keycloak server.



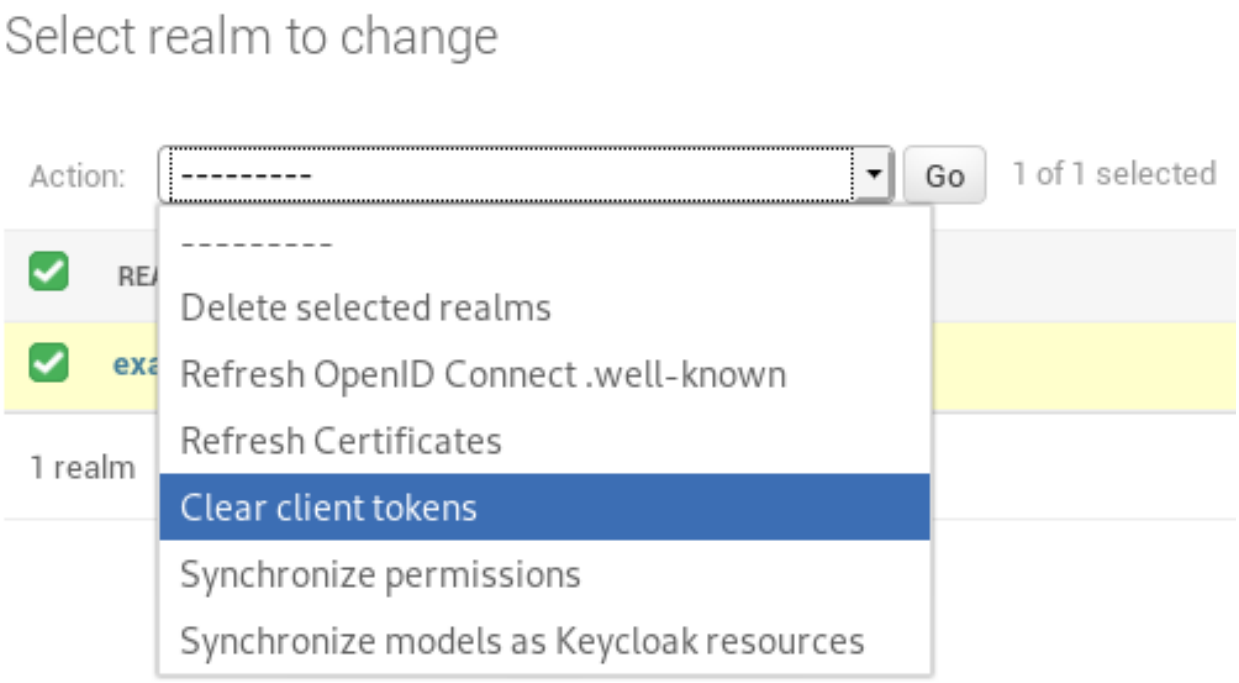
4.3.2 Refresh Certificates

This refreshes the cached certificates from the Keycloak server. These certificates are used for valiation of the JWT's.



4.3.3 Clear client tokens

While debugging client service account permissions it's sometimes required to refresh te session in order to fetch the new permissions. This can be done with this action in the Django admin.



CHAPTER 5

Migrating from Django Auth to Keycloak

There are some tools available which can help by migrating a running project to Keycloak.

5.1 Add user

A management command is available to create a Keycloak user based on a local one.

```
$ python manage.py keycloak_add_user --realm <insert realm name> --username <insert␣
↪user name>
```

Note: In theory it would be possible to synchronize (hashed) passwords to Keycloak however Keycloak uses a 512 bit hash for pbkdf2_sha256 hashed passwords, Django generates a 256 bits hash. In that way passwords will not work when they are copied to Keycloak. The project includes a sha512 hasher (`django_keycloak.hashers.PBKDF2SHA512PasswordHasher`) which you can configure to hash passwords in a Keycloak-complient way.

```
# your-project/settings.py
PASSWORD_HASHERS = [
    'django_keycloak.hashers.PBKDF2SHA512PasswordHasher',
]
```

CHAPTER 6

Permissions by roles

There are two ways of using permissions one by roles and the other one by resources/scopes. The roles method is the default one. In this method the available client roles are available as permissions in your Django Project.

Note: Please read synchronize_permissions if you want to synchronize all available permissions in your current project to roles in Keycloak.

6.1 Setup

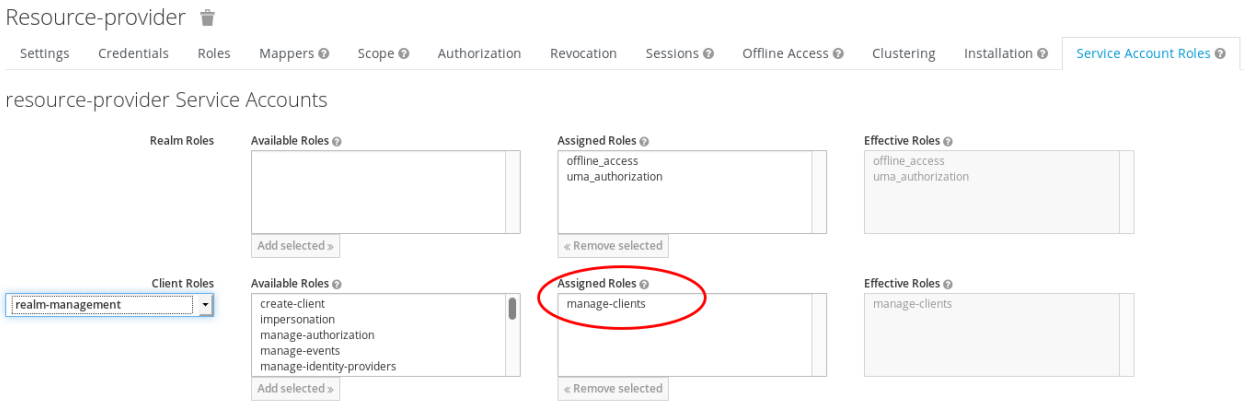
Since this is the default method of handling permission you don't have to configure anything. However it's good to know that the *KEYCLOAK_PERMISSIONS_METHOD* is used to configure the way how permissions are interpreted.

```
# your-project/settings.py
KEYCLOAK_PERMISSIONS_METHOD = 'role'
```

6.2 Synchronize

This Django Admin action which can be triggered for a realm synchronizes all available permission to Keycloak. In keycloak the permissions will get registered as roles. These roles can be added to a user.

For this feature the service account should have the realm-management/manage-clients role assigned.



This only makes sense when you use the *roles* permission method. You can read about this at scenario: *Permissions by roles*.

CHAPTER 7

Permissions by resources and scopes

Next to *Permissions by roles* you can also implement permissions by syncing Django models as resources in Keycloak and the default permissions in Django as scopes in Keycloak.

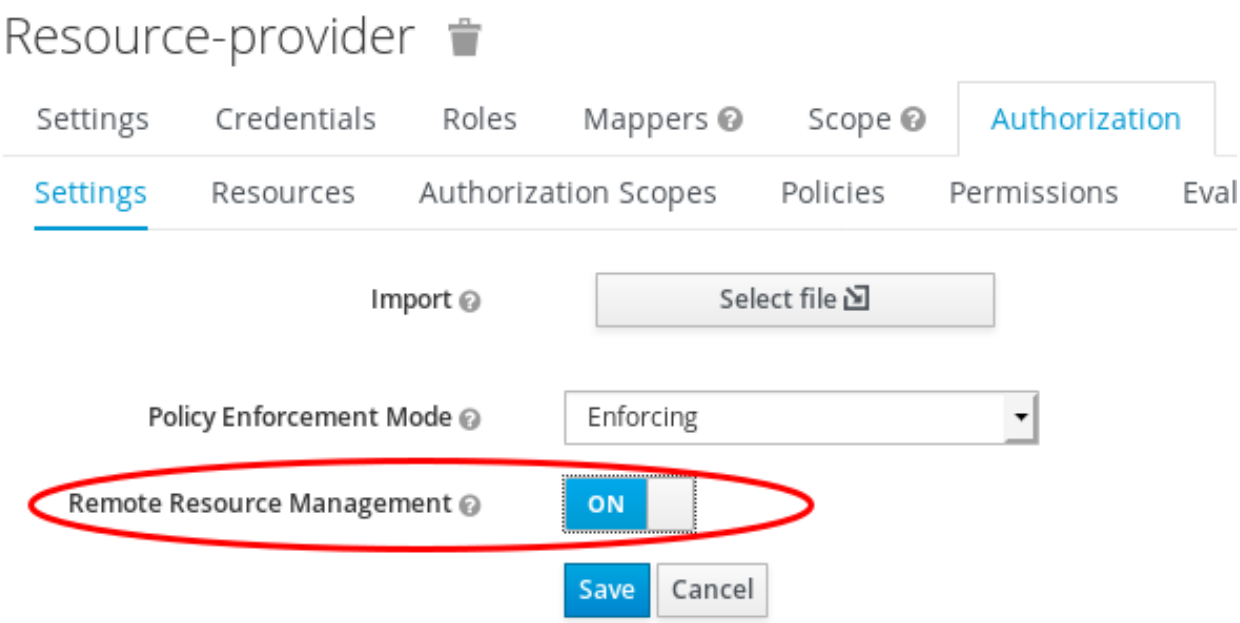
7.1 Setup

To configure Django Keycloak to make use of the Resource / Scope method of permission assigning add the following setting:

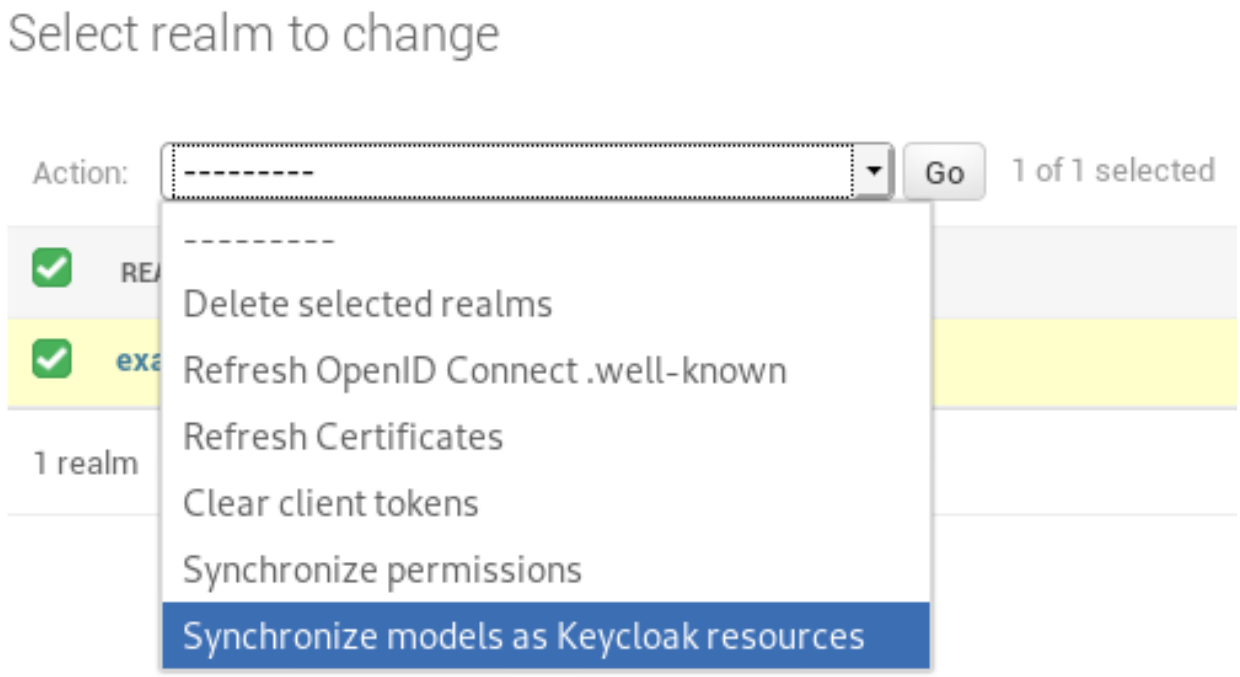
```
# your-project/settings.py
KEYCLOAK_PERMISSIONS_METHOD = 'resource'
```

7.2 Synchronization

In Keycloak enable "Remote Resource Management" for the client:



You can use the Django Admin action "Synchronize models as Keycloak resources" to synchronize models and scopes to Keycloak.



An alternative is to run the Django management command *keycloak_sync_resources*:

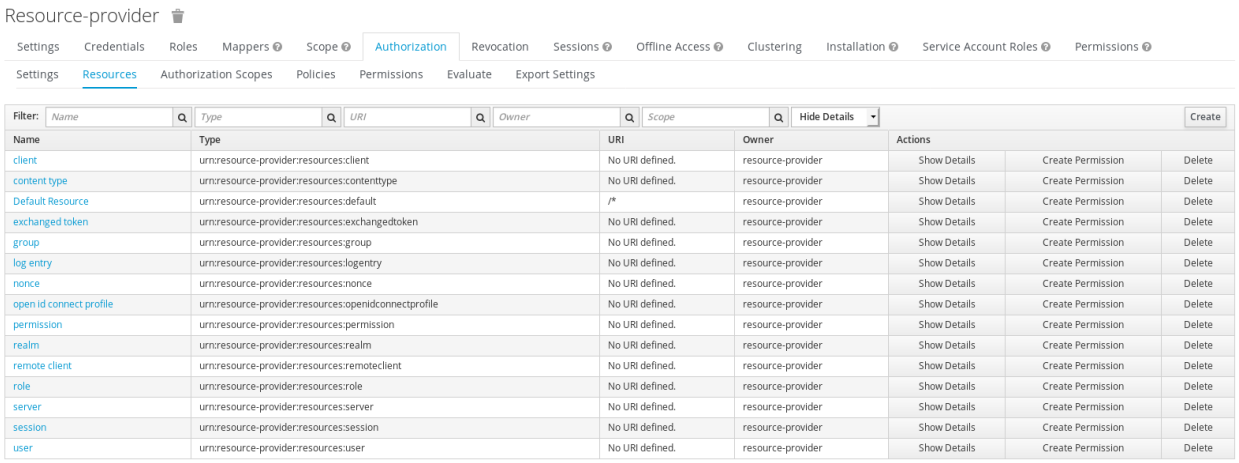
```
$ python manage.py keycloak_sync_resources
```

Optionally you can supply a client to which the resources should be synchronized.

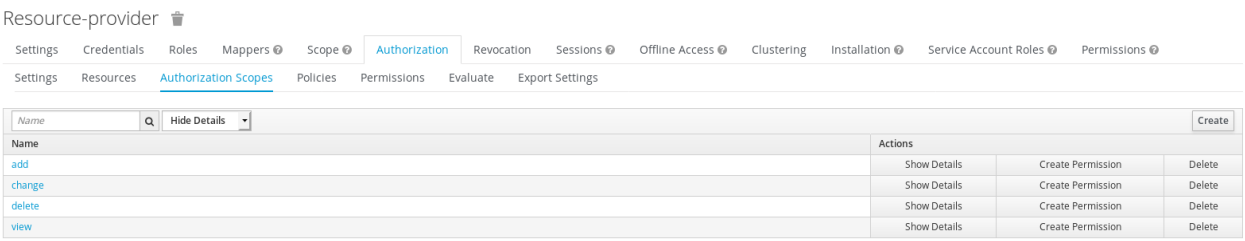
7.3 Usage

After synchronizing you can find the the models as resources and the default permissions as scopes:

Resources:



Scopes:



From here you are able to configure your *policies* and *permissions* and assign them to *users* of *groups* using *roles* in Keycloak. Once assigned you get them back as permissions in Django where the policies are combined with the resources just like you are used to in the default Django permission system i.e. *foo.add_bar* or *foo.change_bar*.

CHAPTER 8

Multi-tenancy

Django Keycloak supports multi tenancy by supporting multiple realms. The way to determine the currently active realm is set to the request in the middleware. In the project there is currently one middleware available. This is *django_keycloak.middleware.BaseKeycloakMiddleware*. This middleware add the first found Realm model to the request object.

```
# your-project/settings.py
MIDDLEWARE = [
    ...

    'django_keycloak.middleware.BaseKeycloakMiddleware',
]
```

If you want to support multiple reams you have to create your own middleware. There are several methods to determine the currently active realm. You can think of realm determination by:

- Hostname
- Environment variable
- Selection during login
- etc.

It's up to you how the realm get determined and therefore it's also up to you to write a proper middleware for it. The only think the middleware has to make the correct Realm model to the request as *request.realm*. This middleware has to be configured above other middlewares which have to be configured for authentication purposes.

Django Keycloak adds Keycloak support to your Django project. It's build on top of Django's authentication system. It works side-by-side with the standard Django authentication implementation and has tools to migrate your current users and permissions to Keycloak.

CHAPTER 9

Features

- Multi tenancy support
- Permissions by roles or by resource/scope
- Choose if you want to create a local User model for every logged in identity or not.

Read *Testing with the Example project* to quickly test this project.

Note: The documentation and the example project are all based on Keycloak version 3.4 since that is the latest version which is commercially supported by Red Hat (SSO).

CHAPTER 10

Installation

Install requirement.

```
$ pip install git+https://github.com/Peter-Slump/django-keycloak.git
```

CHAPTER 11

Setup

Some settings are always required and some other settings are dependant on how you want to integrate Keycloak in your project.

Add *django-keycloak* to your installed apps, add the authentication back-end, add the middleware, configure the urls and point to the correct login page.

```
# your-project/settings.py
INSTALLED_APPS = [
    ....

    'django_keycloak.apps.KeycloakAppConfig'
]

MIDDLEWARE = [
    ...

    'django_keycloak.middleware.BaseKeycloakMiddleware',
]

AUTHENTICATION_BACKENDS = [
    ...

    'django_keycloak.auth.backends.KeycloakAuthorizationCodeBackend',
]

LOGIN_URL = 'keycloak_login'
```

```
# your-project/urls.py
...

urlpatterns = [
    ...
```

(continues on next page)

(continued from previous page)

```
    url(r'^keycloak/', include('django_keycloak.urls')),
]
```

Before you actually start using Django Keycloak make an educated choice between *Setup for local user storage* and *Setup for remote user*.

Then walk through the *Initial setup* to found out how to link your Keycloak instance to your Django project.

If you don't want to take all that effort please read about *Testing with the Example project*

CHAPTER 12

Usage

For requiring a logged in user you can just use the standard Django functionality. This also counts for enforcing permissions.

This app makes use of the Python Keycloak client