djoser Documentation

Release 2.0.1

Piotr Szpetkowski

Dec 31, 2020

Basics

Note: djoser 2.x is not backward compatible with djoser 1.x

CHAPTER 1

Introduction

REST implementation of Django authentication system. **djoser** library provides a set of Django Rest Framework views to handle basic actions such as registration, login, logout, password reset and account activation. It works with custom user model.

Instead of reusing Django code (e.g. `PasswordResetForm`), we reimplemented few things to fit better into Single Page App architecture.

Developed by SUNSCRAPERS with passion & patience.

CHAPTER 2

Getting started

2.1 Available endpoints

- `/users/`
- `/users/me/`
- `/users/confirm/`
- `/users/resend_activation/`
- `/users/set_password/`
- `/users/reset_password/`
- `/users/reset_password_confirm/`
- `/users/set_username/`
- `/users/reset_username/`
- `/users/reset_username_confirm/`
- `/token/login/` (Token Based Authentication)
- `/token/logout/` (Token Based Authentication)
- `/jwt/create/` (JSON Web Token Authentication)
- `/jwt/refresh/` (JSON Web Token Authentication)
- `/jwt/verify/` (JSON Web Token Authentication)

2.2 Supported authentication backends

- Token based authentication from DRF
- JSON Web Token authentication from django-rest-framework-simplejwt

2.3 Supported Python versions

- Python 3.5
- Python 3.6
- Python 3.7
- Python 3.8

2.4 Supported Django versions

- Django 1.11
- Django 2.2
- Django 3.1

2.5 Supported Django Rest Framework versions

- Django Rest Framework 3.9
- Django Rest Framework 3.10
- Django Rest Framework 3.11

2.6 Installation

```
$ pip install -U djoser
```

If you are going to use JWT authentication, you will also need to install djangorestframework_simplejwt with:

```
$ pip install -U djangorestframework_simplejwt
```

Finally if you are going to use third party based authentication e.g. facebook, you will need to install social-auth-app-django with:

```
$ pip install -U social-auth-app-django
```

2.7 Configuration

Configure `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    'django.contrib.auth',
    (...),
    'rest_framework',
    'djoser',
    (...),
)
```

Configure `urls.py`:

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

HTTP Basic Auth strategy is assumed by default as Django Rest Framework does it. We strongly discourage and do not provide any explicit support for basic auth. You should customize your authentication backend as described in *Authentication Backends*.

In case of third party based authentication PSA backend docs will be a great reference to configure given provider.

CHAPTER 3

Sample usage

We provide a standalone test app for you to start easily, see how everything works with basic settings. It might be useful before integrating **djoser** into your backend application.

In this extremely short tutorial we are going to mimic the simplest flow: register user, log in and log out. We will also check resource access on each consecutive step. Let's go!

Clone repository and install **djoser** to your virtualenv:

```
$ git clone git@github.com:sunscrapers/djoser.git
$ cd djoser
$ pip install -e .
```

Go to the `testproject` directory, migrate the database and start the development server:

```
$ cd testproject
$ ./manage.py migrate
$ ./manage.py runserver 8088
```

Register a new user:

```
$ curl -X POST http://127.0.0.1:8088/auth/users/ --data 'username=djoser&
↪password=alpine12'
{"email": "", "username": "djoser", "id":1}
```

So far, so good. We have just created a new user using REST API.

Let's access user's details:

```
$ curl -LX GET http://127.0.0.1:8088/auth/users/me/
{"detail": "Authentication credentials were not provided."}
```

As we can see, we cannot access user profile without logging in. Pretty obvious.

Let's log in:

```
curl -X POST http://127.0.0.1:8088/auth/token/login/ --data 'username=djoser&
↪password=alpine12'
{"auth_token": "b704c9fc3655635646356ac2950269f352ea1139"}
```

We have just obtained an authorization token that we may use later in order to retrieve specific resources.

Let's access user's details again:

```
$ curl -LX GET http://127.0.0.1:8088/auth/users/me/
{"detail": "Authentication credentials were not provided."}
```

Access is still forbidden but let's offer the token we obtained:

```
$ curl -LX GET http://127.0.0.1:8088/auth/users/me/ -H 'Authorization: Token␣
↪b704c9fc3655635646356ac2950269f352ea1139'
{"email": "", "username": "djoser", "id": 1}
```

Yay, it works!

Now let's log out:

```
curl -X POST http://127.0.0.1:8088/auth/token/logout/ -H 'Authorization: Token␣
↪b704c9fc3655635646356ac2950269f352ea1139'
```

And try access user profile again:

```
$ curl -LX GET http://127.0.0.1:8088/auth/users/me/ -H 'Authorization: Token␣
↪b704c9fc3655635646356ac2950269f352ea1139'
{"detail": "Invalid token"}
```

As we can see, user has been logged out successfully and the proper token has been removed.

CHAPTER 4

Authentication Backends

Note: Both Token Based and JWT Authentication can coexist at same time. Simply, follow instructions for both authentication methods and it should work.

4.1 Token Based Authentication

Add `'rest_framework.authtoken'` to `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    'django.contrib.auth',
    (...),
    'rest_framework',
    'rest_framework.authtoken',
    'djoser',
    (...),
]
```

Configure `urls.py`. Pay attention to `djoser.url.authtoken` module path:

```
urlpatterns = [
    (...),
    url(r'^auth/', include('djoser.urls')),
    url(r'^auth/', include('djoser.urls.authtoken')),
]
```

Add `rest_framework.authentication.TokenAuthentication` to Django REST Framework authentication strategies tuple:

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework.authentication.TokenAuthentication',
```

(continues on next page)

(continued from previous page)

```
        (...)
    ),
}
```

Run migrations - this step will create tables for `auth` and `authtoken` apps:

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

4.2 JSON Web Token Authentication

4.2.1 Django Settings

Add `rest_framework_simplejwt.authentication.JWTAuthentication` to Django REST Framework authentication strategies tuple:

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
        (...)
    ),
}
```

Configure *django-rest-framework-simplejwt* to use the *Authorization: JWT <access_token>* header:

```
SIMPLE_JWT = {
   'AUTH_HEADER_TYPES': ('JWT',),
}
```

4.2.2 urls.py

Configure `urls.py` with `djoser.url.jwt` module path:

```
urlpatterns = [
    (...),
    url(r'^auth/', include('djoser.urls')),
    url(r'^auth/', include('djoser.urls.jwt')),
]
```

CHAPTER 5

Settings

You can provide `DJOSER` settings like this:

```
DJOSER = {
    'PASSWORD_RESET_CONFIRM_URL': '#/password/reset/confirm/{uid}/{token}',
    'USERNAME_RESET_CONFIRM_URL': '#/username/reset/confirm/{uid}/{token}',
    'ACTIVATION_URL': '#/activate/{uid}/{token}',
    'SEND_ACTIVATION_EMAIL': True,
    'SERIALIZERS': {},
}
```

Note: All following setting names written in CAPS are keys on `DJOSER` dict.

5.1 USER_ID_FIELD

Name of a unique field in User model to be used as id for `/users/<id>/` endpoints. This is useful if you want to not change default primary key of the User model and hide from public.

Default: `User._meta.pk.name` where `User` is the model set with Django's setting AUTH_USER_MODEL.

5.2 LOGIN_FIELD

Name of a field in User model to be used as login field. This is useful if you want to change the login field from `username` to `email` without providing custom User model.

Default: `User.USERNAME_FIELD` where `User` is the model set with Django's setting AUTH_USER_MODEL.

Warning: Djoser uses djangorestframework-simplejwt to provide a convenient integration for JWT. If you want to change *LOGIN_FIELD* for JWT you need to rely on their documentation and their settings as it's another library. Changing Djoser setting doesn't affect JWT resources.

5.3 PASSWORD_RESET_CONFIRM_URL

URL to your frontend password reset page. It should contain `{uid}` and `{token}` placeholders, e.g. `#/password-reset/{uid}/{token}`. You should pass `uid` and `token` to reset password confirmation endpoint.

Required: `True`

5.4 USERNAME_RESET_CONFIRM_URL

URL to your frontend username reset page. It should contain `{uid}` and `{token}` placeholders, e.g. `#/username-reset/{uid}/{token}`. You should pass `uid` and `token` to reset username confirmation endpoint.

Required: `True`

5.5 SEND_ACTIVATION_EMAIL

If `True` user will be required to click activation link sent in email after:

- creating an account
- updating their email

Default: `False`

5.6 SEND_CONFIRMATION_EMAIL

If `True`, register or activation endpoint will send confirmation email to user.

Default: `False`

5.7 PASSWORD_CHANGED_EMAIL_CONFIRMATION

If `True`, change password endpoints will send confirmation email to user.

Default: `False`

5.8 USERNAME_CHANGED_EMAIL_CONFIRMATION

If `True`, change username endpoints will send confirmation email to user.

Default: `False`

5.9 ACTIVATION_URL

URL to your frontend activation page. It should contain `{uid}` and `{token}` placeholders, e.g. `#/activate/{uid}/{token}`. You should pass `uid` and `token` to activation endpoint.

Required: `True`

5.10 USER_CREATE_PASSWORD_RETYPE

If `True`, you need to pass `re_password` to `/users/` endpoint, to validate password equality.

Default: `False`

5.11 SET_USERNAME_RETYPE

If `True`, you need to pass `re_new_username` to `/users/set_username/` endpoint, to validate username equality.

Default: `False`

5.12 SET_PASSWORD_RETYPE

If `True`, you need to pass `re_new_password` to `/users/set_password/` endpoint, to validate password equality.

Default: `False`

5.13 PASSWORD_RESET_CONFIRM_RETYPE

If `True`, you need to pass `re_new_password` to `/users/reset_password_confirm/` endpoint in order to validate password equality.

Default: `False`

5.14 USERNAME_RESET_CONFIRM_RETYPE

If `True`, you need to pass `re_new_username` to `/users/reset_username_confirm/` endpoint in order to validate username equality.

Default: `False`

5.15 LOGOUT_ON_PASSWORD_CHANGE

If `True`, setting new password will logout the user.

Default: `False`

Note: Logout only works with token based authentication.

5.16 PASSWORD_RESET_SHOW_EMAIL_NOT_FOUND

If `True`, posting a non-existent `email` to `/users/reset_password/` will return a `HTTP_400_BAD_REQUEST` response with an `EMAIL_NOT_FOUND` error message ('User with given email does not exist.').

If `False` (default), the `/users/reset_password/` endpoint will always return a `HTTP_204_NO_CONTENT` response.

Please note that setting this to `True` will expose information whether an email is registered in the system.

Default: `False`

5.17 USERNAME_RESET_SHOW_EMAIL_NOT_FOUND

If `True`, posting a non-existent `email` to `/users/reset_username/` will return a `HTTP_400_BAD_REQUEST` response with an `EMAIL_NOT_FOUND` error message ('User with given email does not exist.').

If `False` (default), the `/users/reset_username/` endpoint will always return a `HTTP_204_NO_CONTENT` response.

Please note that setting this to `True` will expose information whether an email is registered in the system.

Default: `False`

5.18 TOKEN_MODEL

Points to which token model should be used for authentication. In case if only stateless tokens (e.g. JWT) are used in project it should be set to `None`.

Example: `'knox.models.AuthToken'`

Default: `'rest_framework.authtoken.models.Token'`

5.19 SERIALIZERS

Dictionary which maps djoser serializer names to serializer classes (use dotted path). This setting provides a way to easily override given serializer(s) - it is used to update the defaults, so by providing, e.g. one key, all the others will stay default.

Note: Key `'user'` is used for general users whereas `'current_user'` lets you set serializer for special `/users/me` endpoint. They both default to the same serializer though.

Examples

```
{
    'user': 'myapp.serializers.SpecialUserSerializer',
}
```

Default:

```
{
    'activation': 'djoser.serializers.ActivationSerializer',
    'password_reset': 'djoser.serializers.SendEmailResetSerializer',
    'password_reset_confirm': 'djoser.serializers.PasswordResetConfirmSerializer',
    'password_reset_confirm_retype': 'djoser.serializers.
↪PasswordResetConfirmRetypeSerializer',
    'set_password': 'djoser.serializers.SetPasswordSerializer',
    'set_password_retype': 'djoser.serializers.SetPasswordRetypeSerializer',
    'set_username': 'djoser.serializers.SetUsernameSerializer',
    'set_username_retype': 'djoser.serializers.SetUsernameRetypeSerializer',
    'username_reset': 'djoser.serializers.SendEmailResetSerializer',
    'username_reset_confirm': 'djoser.serializers.UsernameResetConfirmSerializer',
    'username_reset_confirm_retype': 'djoser.serializers.
↪UsernameResetConfirmRetypeSerializer',
    'user_create': 'djoser.serializers.UserCreateSerializer',
    'user_create_password_retype': 'djoser.serializers.
↪UserCreatePasswordRetypeSerializer',
    'user_delete': 'djoser.serializers.UserDeleteSerializer',
    'user': 'djoser.serializers.UserSerializer',
    'current_user': 'djoser.serializers.UserSerializer',
    'token': 'djoser.serializers.TokenSerializer',
    'token_create': 'djoser.serializers.TokenCreateSerializer',
}
```

5.20 EMAIL

Dictionary which maps djoser email names to paths to email classes. Same as in case of `SERIALIZERS` it allows partial override.

Examples

```
{
    'activation': 'myapp.email.AwesomeActivationEmail',
}
```

Default:

```
{
    'activation': 'djoser.email.ActivationEmail',
    'confirmation': 'djoser.email.ConfirmationEmail',
    'password_reset': 'djoser.email.PasswordResetEmail',
    'password_changed_confirmation': 'djoser.email.PasswordChangedConfirmationEmail',
    'username_changed_confirmation': 'djoser.email.UsernameChangedConfirmationEmail',
    'username_reset': 'djoser.email.UsernameResetEmail',
}
```

5.21 CONSTANTS

Dictionary which maps djoser constant names to paths to constant classes. Same as in case of `SERIALIZERS` it allows partial override.

Examples

```
{
    'messages': 'myapp.constants.CustomMessages',
}
```

Default:

```
{
    'messages': 'djoser.constants.Messages',
}
```

5.22 SOCIAL_AUTH_TOKEN_STRATEGY

String path to class responsible for token strategy used by social authentication.

Example: `'myapp.token.MyStrategy'`

Default: `'djoser.social.token.jwt.TokenStrategy'`

5.23 SOCIAL_AUTH_ALLOWED_REDIRECT_URIS

List of allowed redirect URIs for social authentication.

Example: `['https://auth.example.com']`

Default: `[]`

5.24 PERMISSIONS

Changed in version 2.0.

Dictionary that maps permissions to certain views across Djoser.

Note: `Admin` in class names refers to users that have `is_staff` flag set to True, not superusers.

Examples

```
{
    'user': ['djoser.permissions.CurrentUserOrAdminOrReadOnly']
}
```

Defaults

```
{
    'activation': ['rest_framework.permissions.AllowAny'],
    'password_reset': ['rest_framework.permissions.AllowAny'],
    'password_reset_confirm': ['rest_framework.permissions.AllowAny'],
    'set_password': ['djoser.permissions.CurrentUserOrAdmin'],
    'username_reset': ['rest_framework.permissions.AllowAny'],
    'username_reset_confirm': ['rest_framework.permissions.AllowAny'],
    'set_username': ['djoser.permissions.CurrentUserOrAdmin'],
    'user_create': ['rest_framework.permissions.AllowAny'],
    'user_delete': ['djoser.permissions.CurrentUserOrAdmin'],
    'user': ['djoser.permissions.CurrentUserOrAdmin'],
    'user_list': ['djoser.permissions.CurrentUserOrAdmin'],
    'token_create': ['rest_framework.permissions.AllowAny'],
    'token_destroy': ['rest_framework.permissions.IsAuthenticated'],
}
```

5.25 HIDE_USERS

New in version 2.0.

If set to True, listing `/users/` enpoint by normal user will return only that user's profile in the list. Beside that, accessing `/users/<id>/` endpoints by user without proper permission will result in HTTP 404 instead of HTTP 403.

Default: `True`

CHAPTER 6

Base Endpoints

6.1 User Create

Use this endpoint to register new user. Your user model manager should implement create_user method and have USERNAME_FIELD and REQUIRED_FIELDS fields.

Default URL: `/users/`

Note: `re_password` is only required if `USER_CREATE_PASSWORD_RETYPE` is `True`

Method	Request	Response

`POST`	• `{{ User.USERNAME_FIELD }}`
	• `{{ User.REQUIRED_FIELDS }}`
	• `password`
	• `re_password`
		`HTTP_201_CREATED`
		• `{{ User.USERNAME_FIELD }}`
		• `{{ User._meta.pk.name }}`
		• `{{ User.REQUIRED_FIELDS }}`
		`HTTP_400_BAD_REQUEST`
		• `{{ User.USERNAME_FIELD }}`
		• `{{ User.REQUIRED_FIELDS }}`
		• `password`
		• `re_password`

6.2 User Activate

Use this endpoint to activate user account. This endpoint is not a URL which will be directly exposed to your users - you should provide site in your frontend application (configured by `ACTIVATION_URL`) which will send `POST` request to activate endpoint. `HTTP_403_FORBIDDEN` will be raised if user is already active when calling this endpoint (this will happen if you call it more than once).

Default URL: `/users/activation/`

Method	Request	Response

`POST`	• `uid`
	• `token`
		`HTTP_204_NO_CONTENT`
		`HTTP_400_BAD_REQUEST`
		• `uid`
		• `token`
		`HTTP_403_FORBIDDEN`
		• `detail`

6.3 User Resend Activation E-mail

Use this endpoint to re-send the activation e-mail. Note that no e-mail would be sent if the user is already active or if they don't have a usable password. Also if the sending of activation e-mails is disabled in settings, this call will result in `HTTP_400_BAD_REQUEST`

Default URL: `/users/resend_activation/`

Method	Request	Response
`POST`	• `{{ User.EMAIL_FIELD }}`	`HTTP_204_NO_CONTENT` `HTTP_400_BAD_REQUEST`

6.4 User

Use this endpoint to retrieve/update the authenticated user.

Default URL: `/users/me/`

Method	Request	Response
GET	–	HTTP_200_OK • {{ User.USERNAME_FIELD }} • {{ User._meta.pk.name }} • {{ User.REQUIRED_FIELDS }}
PUT	{{ User.REQUIRED_FIELDS }}	HTTP_200_OK • {{ User.USERNAME_FIELD }} • {{ User._meta.pk.name }} • {{ User.REQUIRED_FIELDS }} HTTP_400_BAD_REQUEST • {{ User.REQUIRED_FIELDS }}
PATCH	{{ User.FIELDS_TO_UPDATE }}	HTTP_200_OK • {{ User.USERNAME_FIELD }} • {{ User._meta.pk.name }} • {{ User.REQUIRED_FIELDS }} HTTP_400_BAD_REQUEST • {{ User.REQUIRED_FIELDS }}

6.5 User Delete

Use this endpoint to delete authenticated user. By default it will simply verify password provided in `current_password`, delete the auth token if token based authentication is used and invoke delete for a given `User` instance. One of ways to customize the delete behavior is to override `User.delete`.

Default URL: `/users/me/`

Method	Request	Response
DELETE	• current_password	HTTP_204_NO_CONTENT HTTP_400_BAD_REQUEST • current_password

6.6 Set Username

Use this endpoint to change user's USERNAME_FIELD. By default this changes the username.

Note: URLs of following settings rely on the User model. Django allows you to set User.USERNAME_FIELD and User.EMAIL_FIELD fields and Djoser respects that by modifying its default url structure and serializers to reflect that settings. When you see {USERNAME_FIELD} or {EMAIL_FIELD} in the settings below, it means that those parts will be substituted with what you set in your User model.

For example: here, the default URL is presented like this: /users/set_{USERNAME_FIELD}/ this means that if your custom User model has USERNAME_FIELD set to nickname, the URL will look like this: /users/set_nickname/. The same rule applies to fields sent with the request.

Default URL: /users/set_{USERNAME_FIELD}/

Note: re_new_{USERNAME_FIELD} is only required if SET_USERNAME_RETYPE is True

Method	Request	Response
POST	• new_{USERNAME_FIELD} • re_new_{USERNAME_FIELD} • current_password	HTTP_204_NO_CONTENT HTTP_400_BAD_REQUEST • new_{USERNAME_FIELD} • re_new_{USERNAME_FIELD} • current_password

6.7 Reset Username

Use this endpoint to send email to user with username reset link. You have to setup USERNAME_RESET_CONFIRM_URL.

Default URL: /users/reset_{USERNAME_FIELD}/

Note: HTTP_204_NO_CONTENT if USERNAME_RESET_SHOW_EMAIL_NOT_FOUND is False

Otherwise if the value of {EMAIL_FIELD} does not exist in database HTTP_400_BAD_REQUEST

Method	Request	Response
POST	{EMAIL_FIELD}	HTTP_204_NO_CONTENT HTTP_400_BAD_REQUEST • {EMAIL_FIELD}

6.8 Reset Username Confirmation

Use this endpoint to finish reset username process. This endpoint is not a URL which will be directly exposed to your users - you should provide site in your frontend application (configured by `USERNAME_RESET_CONFIRM_URL`) which will send `POST` request to reset username confirmation endpoint. `HTTP_400_BAD_REQUEST` will be raised if the user has logged in or changed username since the token creation.

Default URL: `/users/reset_{USERNAME_FIELD}_confirm/`

Note: `re_new_username` is only required if `USERNAME_RESET_CONFIRM_RETYPE` is `True`

Method	Request	Response

`POST`	• `uid`
	• `token`
	• `new_{USERNAME_FIELD}`
	• `re_new_{USERNAME_FIELD}`
		`HTTP_204_NO_CONTENT`
		`HTTP_400_BAD_REQUEST`
		• `uid`
		• `token`
		• `new_{USERNAME_FIELD}`
		• `re_new_{USERNAME_FIELD}`

6.9 Set Password

Use this endpoint to change user password.

Default URL: `/users/set_password/`

Note: `re_new_password` is only required if `SET_PASSWORD_RETYPE` is `True`

Method	Request	Response

`POST`	• `new_password`
	• `re_new_password`
	• `current_password`
		`HTTP_204_NO_CONTENT`
		`HTTP_400_BAD_REQUEST`
		• `new_password`
		• `re_new_password`
		• `current_password`

6.10 Reset Password

Use this endpoint to send email to user with password reset link. You have to setup `PASSWORD_RESET_CONFIRM_URL`.

Default URL: `/users/reset_password/`

Note: `HTTP_204_NO_CONTENT` if `PASSWORD_RESET_SHOW_EMAIL_NOT_FOUND` is `False`

Otherwise if the value of `{EMAIL_FIELD}` does not exist in database `HTTP_400_BAD_REQUEST`

Method	Request	Response

`POST`	`{EMAIL_FIELD}`	`HTTP_204_NO_CONTENT`
		`HTTP_400_BAD_REQUEST`
		• `{EMAIL_FIELD}`

6.11 Reset Password Confirmation

Use this endpoint to finish reset password process. This endpoint is not a URL which will be directly exposed to your users - you should provide site in your frontend application (configured by `PASSWORD_RESET_CONFIRM_URL`) which will send `POST` request to reset password confirmation endpoint. `HTTP_400_BAD_REQUEST` will be raised if the user has logged in or changed password since the token creation.

Default URL: `/users/reset_password_confirm/`

Note: `re_new_password` is only required if `PASSWORD_RESET_CONFIRM_RETYPE` is `True`

Method	Request	Response

`POST`	• `uid`
	• `token`
	• `new_password`
	• `re_new_password`
		`HTTP_204_NO_CONTENT`
		`HTTP_400_BAD_REQUEST`
		• `uid`
		• `token`
		• `new_password`
		• `re_new_password`

CHAPTER 7

Token Endpoints

7.1 Token Create

Use this endpoint to obtain user authentication token. This endpoint is available only if you are using token based authentication.

Default URL: `/token/login/`

Method	Request	Response
`POST`	• `{{ User.USERNAME_FIELD }}` • `password`	`HTTP_200_OK` • `auth_token`

7.2 Token Destroy

Use this endpoint to logout user (remove user authentication token). This endpoint is available only if you are using token based authentication.

Default URL: `/token/logout/`

Method	Request	Response
`POST`	–	`HTTP_204_NO_CONTENT`

CHAPTER 8

JWT Endpoints

Note: Djoser settings won't have an effect on your JWT resources. Visit djangorestframework-simplejwt to check what can be configured.

8.1 JWT Create

Use this endpoint to obtain JWT.

Default URL: `/jwt/create/`

Method	Request	Response

`POST`	• `{{ User.USERNAME_FIELD }}`
	• `password`
		`HTTP_200_OK`
		• `access`
		• `refresh`
		`HTTP_401_UNAUTHORIZED`
		• `non_field_errors`

8.2 JWT Refresh

Use this endpoint to refresh JWT.

Default URL: `/jwt/refresh/`

Method	Request	Response
POST	• refresh	HTTP_200_OK • access HTTP_401_UNAUTHORIZED • non_field_errors

8.3 JWT Verify

Use this endpoint to verify JWT.

Default URL: /jwt/verify/

Method	Request	Response
POST	• token	HTTP_200_OK HTTP_401_UNAUTHORIZED • non_field_errors

CHAPTER 9

Social Endpoints

Warning: This API is in beta quality - backward compatibility is not guaranteed in future versions and you may come across bugs.

9.1 Provider Auth

Using these endpoints you can authenticate with external tools.

The workflow should look like this:

1. Access the endpoint providing a `redirect_uri` that would perform the `POST` action later.
2. The request would return a JSON containing one key `authorization_url`. Redirect the user to that URL.
3. When the user authenticates with the external tool, that tool would redirect them to the `redirect_uri` you provided with a `GET` querystring containing two arguments: `code` and `state`
4. From the view that your user got redirected to, issue a `POST` request to the endpoint with the `code` and `state` arguments. You should use `application/x-www-form-urlencoded` not JSON. The user should be now authenticated in your application.

The list of providers is available at social backend docs. please follow the instructions provided there to configure your backend.

Configure `urls.py`:

```
urlpatterns = [
    (...),
    url(r'^auth/', include('djoser.social.urls')),
]
```

Default URL: `/o/{{ provider }}/`

Note:

- `redirect_uri` is provided via GET parameters - not JSON
- `state` parameter isn't always required e.g. in case of OpenID backends

Method	Request	Response

`GET`	• `redirect_uri`	`HTTP_200_OK`
		• `authorization_url`
		`HTTP_400_BAD_REQUEST`

`POST`	• `code`
	• `state`
		`HTTP_201_CREATED`
		• `token`
		`HTTP_400_BAD_REQUEST`
		• `non_field_errors`

CHAPTER 10

Signals

Djoser provides a set of signals that allow you to hook into Djoser user management flow.

10.1 user_registered

This signal is sent after successful user registration.

Argument	Value
`sender`	sender class
`user`	user instance
`request`	request instance

At this point, `user` has already been created and saved.

10.2 user_activated

This signal is sent after successful user activation.

Argument	Value
`sender`	sender class
`user`	user instance
`request`	request instance

At this point, `user` has already been activated and saved.

CHAPTER 11

Migration Guide

11.1 Migrating from 1.x to 2.0

Here are some advices to help you with the transition to new Djoser.

1. If you still use Python 2.x - stay on Djoser 1.x.
2. If you still use Django REST Framework 3.9 or lower - stay on Djoser 1.x.
3. There were several changes to default *settings*
4. User-related endpoints are gathered within UserViewSet.

11.1.1 Some View class names and URLs has been updated or removed

View class names:

- `RootView` has been removed
- `UserCreateView`, `UserDeleteView`, `UserView`, `PasswordResetView`,

`SetPasswordView`, `PasswordResetConfirmView`, `SetUsernameView`, `ActivationView`, and `ResendActivationView` have all been removed and replaced by appropriate sub-views within `UserViewSet`.

If you subclassed any of those views, you need to refactor your code - we suggest subclassing UserViewSet and overwrite appropriate methods there.

Base URLs:

- `users/create/`, `users/delete/`, `users/confirm/`, and `users/resend/` removed; use viewset-provided endpoints (see *settings*)
- `password/` has been renamed to `users/set_password/`
- `password/reset/` has been renamed to `users/reset_password/`
- `password/reset/confirm/` has been renamed to `users/reset_password_confirm/`

Token Based Authentication URLs:

- use `token/login` to create token
- user `token/logout` to invalidate the token

Added URLs: * `users/set_{0}/` format(User.USERNAME_FIELD) * `users/reset_{0}/` format(User.USERNAME_FIELD) * `users/reset_{0}_confirm/` format(User.USERNAME_FIELD)

If anything else stopped working: consult *settings* first before filing a bug report.

CHAPTER 12

Indices and tables

- genindex
- search