djoser Documentation

Release 1.1.5

Piotr Szpetkowski

Jul 24, 2018

Basics

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/change_username/`
- `/password/`
- `/password/reset/`
- `/password/reset/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-jwt

2.3 Supported Python versions

- Python 2.7
- Python 3.4
- Python 3.5
- Python 3.6

2.4 Supported Django versions

- Django 1.10
- Django 1.11

2.5 Supported Django Rest Framework versions

- Django Rest Framework 3.7

2.6 Installation

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

If you are going to use JWT authentication, you will also need to install djangorestframework-jwt with:

```
$ pip install -U djangorestframework-jwt
```

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=djoser'
{"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 -X 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=djoser'
{"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 -X 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 -X 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 -X 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

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

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

Add `rest_framework_jwt.authentication.JSONWebTokenAuthentication` to Django REST Framework authentication strategies tuple:

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_jwt.authentication.JSONWebTokenAuthentication',
        (...)
    ),
}
```

CHAPTER 5

Settings

You may optionally provide `DJOSER` settings:

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

5.1 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`

5.2 SEND_ACTIVATION_EMAIL

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

- creating an account via `RegistrationView`
- updating his email via `UserView`

Default: `False`

5.3 SEND_CONFIRMATION_EMAIL

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

Default: `False`

5.4 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`

5.5 SET_USERNAME_RETYPE

If `True`, you need to pass `re_new_{{ User.USERNAME_FIELD }}` to `/{{ User.USERNAME_FIELD }}/` endpoint, to validate username equality.

Default: `False`

5.6 SET_PASSWORD_RETYPE

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

Default: `False`

5.7 PASSWORD_RESET_CONFIRM_RETYPE

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

Default: `False`

5.8 LOGOUT_ON_PASSWORD_CHANGE

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

Default: `False`

5.9 USER_EMAIL_FIELD_NAME

Determines which field in `User` model is used for email in versions of Django before 1.11. In Django 1.11 and greater value of this setting is ignored and value provided by `User.get_email_field_name` is used. This setting will be dropped when Django 1.8 LTS goes EOL.

Default: `'email'`

5.10 PASSWORD_RESET_SHOW_EMAIL_NOT_FOUND

If `True`, posting a non-existent `email` to `/password/reset/` 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 `/password/reset/` 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`

5.11 TOKEN_MODEL

Points to which token model should be used for authentication.

Example: `'knox.models.AuthToken'` **Default**: `'rest_framework.authtoken.models.Token'`

5.12 SERIALIZERS

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

Examples

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

Default:

```
{
    'activation': 'djoser.serializers.ActivationSerializer',
    'password_reset': 'djoser.serializers.PasswordResetSerializer',
    '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',
    'user_create': 'djoser.serializers.UserCreateSerializer',
    'user_delete': 'djoser.serializers.UserDeleteSerializer',
    'user': 'djoser.serializers.UserSerializer',
    'token': 'djoser.serializers.TokenSerializer',
    'token_create': 'djoser.serializers.TokenCreateSerializer',
}
```

5.13 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',
}
```

5.14 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'`

5.15 SOCIAL_AUTH_ALLOWED_REDIRECT_URIS

List of allowed redirect URIs for social authentication.

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

CHAPTER 6

Base Endpoints

6.1 User

Use this endpoint to retrieve/update user.

Default URL: `/users/me/` **Backward-compatible URL**: `/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 }}`

6.2 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/` **Backward-compatible URL**: `/users/create/`

Method	Request	Response
POST	• {{ User.USERNAME_FIELD }} • {{ User.REQUIRED_FIELDS }} • password	HTTP_201_CREATED • {{ User.USERNAME_FIELD }} • {{ User._meta.pk.name }} • {{ User.REQUIRED_FIELDS }}

6.3 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

Backward-compatible URL: `/users/delete/`

Method	Request	Response
POST	• current_password	HTTP_204_NO_CONTENT HTTP_400_BAD_REQUEST • current_password

6.4 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.

Default URL: `/users/confirm/` **Backward-compatible URL**: `/users/activate/`

Method	Request	Response
POST	• uid • token	HTTP_204_NO_CONTENT

6.5 Set Username

Use this endpoint to change user username (USERNAME_FIELD).

Default URL: /users/change_username/ **Backward-compatible URL**: /{{ User.USERNAME_FIELD }}/

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

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

6.6 Set Password

Use this endpoint to change user password.

Default URL: /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

6.7 Reset Password

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

Default URL: /password/reset/

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

Otherwise and if `email` does not exist in database `HTTP_400_BAD_REQUEST`

Method	Request	Response
`POST`	`email`	• `HTTP_204_NO_CONTENT` • `HTTP_400_BAD_REQUEST`

6.8 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.

Default URL: `/password/reset/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`

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/` **Backward-compatible URL**: `/token/create/`

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/` **Backward-compatible URL**: `/token/destroy/`

Method	Request	Response
`POST`	–	`HTTP_204_NO_CONTENT`

CHAPTER 8

JWT Endpoints

8.1 JWT Create

Use this endpoint to obtain JWT.

Default URL: `/jwt/create/`

Method	Request	Response

`POST`	• `token`	`HTTP_200_OK`
		• `token`
		`HTTP_400_BAD_REQUEST`
		• `non_field_errors`

8.2 JWT Refresh

Use this endpoint to refresh JWT.

Default URL: `/jwt/refresh/`

Method	Request	Response

`POST`	• `token`	`HTTP_200_OK`
		• `token`
		`HTTP_400_BAD_REQUEST`
		• `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`
		• `token`
		`HTTP_400_BAD_REQUEST`
		• `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

Use this endpoint to obtain authorization URL for a given provider with the GET method or to obtain authentication token with POST method. List of providers is available at social backend docs.

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

Migration Guide

10.1 Migrating from 1.1 to 1.2

There is no urgent need to change anything as backward compatibility is retained. That being said we ask you to change usage from old endpoints to new ones for the warm fuzzy feeling of being more RESTful :)

10.2 Migrating from 0.x to 1.0

The stable release has introduced a number of backward incompatible changes and purpose of this guide is to allow developer to quickly adapt a given project.

10.2.1 Removal of `UserEmailFactoryBase` and its subclasses

As mentioned in Emails page since 1.0 email support has been removed from Djoser and it is advised to use django-templated-mail for use cases which were previously handled by djoser email support. You can find out more about it in the project documentation. Keep in mind that `DOMAIN` and `SITE_NAME` settings have also been moved to django-templated-mail as described in settings page.

10.2.2 Base URLs are no longer included with other URLs

Previously `djoser.urls.base` were bundled with `djoser.urls.authtoken`, however in some cases developer might not need them and therefore if base URLs are needed it is now necessary to explicitly include them, e.g.:

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

10.2.3 Dropped support for Django < 1.10

Support for Django 1.8 and 1.9 has been dropped in Django REST Framework 3.7 and hence there was no reason to keep it in djoser. It is recommended to upgrade to Django 1.11, since 1.10 will EOL in December 2017. Django Deprecation Timeline and Django Release Notes are very helpful in the process.

10.2.4 Some View class names and URLs has been updated

Also please note that for sake of consistency all URLs now end with a trailing slash. The trailing slash is optional to ensure compatibility with frontend tools that strip the trailing slash (eg Google's Chrome browser and Angular framework).

View class names:

- `RegistrationView` has been renamed to `UserCreateView`
- `LoginView` has been renamed to `TokenCreateView`
- `LogoutView` has been renamed to `TokenDestroyView`

Base URLs:

- `register/` has been renamed to `users/create/`
- `register` URL name has been renamed to `user-create`
- `activate/` has been renamed to `users/activate/`
- `activate` URL name has been renamed to `user-activate`

Token Based Authentication URLs:

- `login/` has been renamed to `token/create/`
- `login` URL name has been renamed to `token-create`
- `logout/` has been renamed to `token/destroy/`
- `logout` URL name has been renamed to `token-destroy`

CHAPTER 11

Emails

Explicit email support has been removed from djoser in 1.0.0. It didn't make sense to handle such responsibility in a package, which should simply provide an implementation of common authentication-related REST endpoints.

Email support is now handled with the django-templated-mail package.

Email classes can be overridden using EMAIL setting

CHAPTER 12

Adjustment

If you need to customize any serializer behaviour you can use the `DJOSER['SERIALIZERS']` setting to use your own serializer classes in the built-in views. Or if you need to completely change the default djoser behaviour, you can always override djoser views with your own custom ones.

Define custom `urls` instead of reusing `djoser.urls`:

```
urlpatterns = patterns('',
    (...),
    url(r'^register/$', views.CustomRegistrationView.as_view()),
)
```

Define custom view/serializer (inherit from one of `djoser` class) and override necessary method/field:

```
class CustomRegistrationView(djoser.views.RegistrationView):

    def send_activation_email(self, *args, **kwargs):
        your_custom_email_sender(*args, **kwargs)
```

You could check `djoser` API in source code:

- djoser.views
- djoser.serializers

CHAPTER 13

Examples

13.1 Early detecting invalid password reset tokens

When there is need to check if password reset token is still valid without actually resetting the password it is possible to approach the problem like so:

```
from django.contrib.auth.tokens import default_token_generator
from rest_framework import generics, permissions, status
from rest_framework.response import Response

from djoser import serializers


class PasswordTokenCheckView(generics.CreateAPIView):
    permission_classes = (
        permissions.AllowAny,
    )
    token_generator = default_token_generator
    serializer_class = serializers.UidAndTokenSerializer

    def post(self, request, *args, **kwargs):
        serializer = self.get_serializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        headers = self.get_success_headers(serializer.data)
        return Response(serializer.data, status=status.HTTP_200_OK, headers=headers)
```

CHAPTER 14

Indices and tables

- genindex
- search