dj-rest-auth Documentation

Release 5.0.1

iMerica Inc.

Sep 17, 2023

Contents

1 Contents **3**

1.1 Introduction . . . 3
1.2 Installation . . . 4
1.3 API endpoints . . . 10
1.4 Configuration . . . 12
1.5 LOGIN_SERIALIZER . . . 13
1.6 TOKEN_SERIALIZER . . . 13
1.7 JWT_SERIALIZER . . . 13
1.8 JWT_SERIALIZER_WITH_EXPIRATION . . . 13
1.9 JWT_TOKEN_CLAIMS_SERIALIZER . . . 13
1.10 USER_DETAILS_SERIALIZER . . . 14
1.11 PASSWORD_RESET_SERIALIZER . . . 14
1.12 PASSWORD_RESET_CONFIRM_SERIALIZER . . . 14
1.13 PASSWORD_CHANGE_SERIALIZER . . . 14
1.14 REGISTER_SERIALIZER . . . 14
1.15 REGISTER_PERMISSION_CLASSES . . . 14
1.16 TOKEN_MODEL . . . 15
1.17 TOKEN_CREATOR . . . 15
1.18 PASSWORD_RESET_USE_SITES_DOMAIN . . . 15
1.19 OLD_PASSWORD_FIELD_ENABLED . . . 15
1.20 LOGOUT_ON_PASSWORD_CHANGE . . . 15
1.21 SESSION_LOGIN . . . 15
1.22 USE_JWT . . . 15
1.23 JWT_AUTH_COOKIE . . . 16
1.24 JWT_AUTH_REFRESH_COOKIE . . . 16
1.25 JWT_AUTH_REFRESH_COOKIE_PATH . . . 16
1.26 JWT_AUTH_SECURE . . . 16
1.27 JWT_AUTH_HTTPONLY . . . 16
1.28 JWT_AUTH_SAMESITE . . . 16
1.29 JWT_AUTH_RETURN_EXPIRATION . . . 16
1.30 JWT_AUTH_COOKIE_USE_CSRF . . . 16
1.31 JWT_AUTH_COOKIE_ENFORCE_CSRF_ON_UNAUTHENTICATED . . . 17
1.32 Demo project . . . 17
1.33 FAQ . . . 17
1.34 Vulnerability Disclosure Policy . . . 18

Note: dj-rest-auth version 1.0.0 now uses Django Simple JWT.

CHAPTER 1

Contents

1.1 Introduction

Since the introduction of django-rest-framework, Django apps have been able to serve up app-level REST API endpoints. As a result, we saw a lot of instances where developers implemented their own REST registration API endpoints here and there, snippets, and so on. We aim to solve this demand by providing dj-rest-auth, a set of REST API endpoints to handle User Registration and Authentication tasks. By having these API endpoints, your client apps such as AngularJS, iOS, Android, and others can communicate to your Django backend site independently via REST APIs for User Management. Of course, we'll add more API endpoints as we see the demand.

1.1.1 Features

- User Registration with activation
- Login/Logout
- Retrieve/Update the Django User model
- Password change
- Password reset via e-mail
- Social Media authentication

1.1.2 Apps structure

- `dj_rest_auth` has basic auth functionality like login, logout, password reset and password change
- `dj_rest_auth.registration` has logic related with registration and social media authentication

1.1.3 Demo projects

- You can also check our *Demo Project* which is using jQuery on frontend.
- There is also a React demo project based on Create-React-App in demo/react-spa/

1.2 Installation

1. Install package:

```
pip install dj-rest-auth
```

2. Add `dj_rest_auth` app to INSTALLED_APPS in your django settings.py:

```
INSTALLED_APPS = (
    ...,
    'rest_framework',
    'rest_framework.authtoken',
    ...,
    'dj_rest_auth'
)
```

Note: This project depends on `django-rest-framework` library, so install it if you haven't done yet. Make sure also you have installed `rest_framework` and `rest_framework.authtoken` apps

3. Add dj_rest_auth urls:

```
urlpatterns = [
    ...,
    path('dj-rest-auth/', include('dj_rest_auth.urls'))
]
```

4. Migrate your database

```
python manage.py migrate
```

You're good to go now!

1.2.1 Registration (optional)

1. If you want to enable standard registration process you will need to install `django-allauth` by using `pip install 'dj-rest-auth[with_social]'`.
2. Add `django.contrib.sites`, `allauth`, `allauth.account`, `allauth.socialaccount` and `dj_rest_auth.registration` apps to INSTALLED_APPS in your django settings.py:
3. Add `SITE_ID = 1` to your django settings.py

```
INSTALLED_APPS = (
    ...,
    'django.contrib.sites',
    'allauth',
    'allauth.account',
```

(continues on next page)

(continued from previous page)

```
    'allauth.socialaccount',
    'dj_rest_auth.registration',
)

SITE_ID = 1
```

3. Add dj_rest_auth.registration urls:

```
urlpatterns = [
    ...,
    path('dj-rest-auth/', include('dj_rest_auth.urls')),
    path('dj-rest-auth/registration/', include('dj_rest_auth.registration.urls'))
]
```

1.2.2 Social Authentication (optional)

Using `django-allauth`, `dj-rest-auth` provides helpful class for creating social media authentication view.

Note: Points 1 and 2 are related to `django-allauth` configuration, so if you have already configured social authentication, then please go to step 3. See `django-allauth` documentation for more details.

1. Add `allauth.socialaccount` and `allauth.socialaccount.providers.facebook` or `allauth.socialaccount.providers.twitter` apps to INSTALLED_APPS in your django settings.py:

```
INSTALLED_APPS = (
    ...,
    'rest_framework',
    'rest_framework.authtoken',
    'dj_rest_auth'
    ...,
    'django.contrib.sites',
    'allauth',
    'allauth.account',
    'dj_rest_auth.registration',
    ...,
    'allauth.socialaccount',
    'allauth.socialaccount.providers.facebook',
    'allauth.socialaccount.providers.twitter',

)
```

2. Add Social Application in django admin panel

Facebook

3. Create new view as a subclass of `dj_rest_auth.registration.views.SocialLoginView` with `FacebookOAuth2Adapter` adapter as an attribute:

```
from allauth.socialaccount.providers.facebook.views import FacebookOAuth2Adapter
from dj_rest_auth.registration.views import SocialLoginView
```

(continues on next page)

(continued from previous page)

```
class FacebookLogin(SocialLoginView):
    adapter_class = FacebookOAuth2Adapter
```

4. Create url for FacebookLogin view:

```
urlpatterns += [
    ...,
    path('dj-rest-auth/facebook/', FacebookLogin.as_view(), name='fb_login')
]
```

Twitter

If you are using Twitter for your social authentication, it is a bit different since Twitter uses OAuth 1.0.

3. Create new view as a subclass of `dj_rest_auth.registration.views.SocialLoginView` with `TwitterOAuthAdapter` adapter and `TwitterLoginSerializer` as an attribute:

```
from allauth.socialaccount.providers.twitter.views import TwitterOAuthAdapter
from dj_rest_auth.registration.views import SocialLoginView
from dj_rest_auth.social_serializers import TwitterLoginSerializer

class TwitterLogin(SocialLoginView):
    serializer_class = TwitterLoginSerializer
    adapter_class = TwitterOAuthAdapter
```

4. Create url for TwitterLogin view:

```
urlpatterns += [
    ...,
    path('dj-rest-auth/twitter/', TwitterLogin.as_view(), name='twitter_login')
]
```

Note: Starting from v0.21.0, django-allauth has dropped support for context processors. Check out http://django-allauth.readthedocs.org/en/latest/changelog.html#from-0-21-0 for more details.

GitHub

If you are using GitHub for your social authentication, it uses code and not AccessToken directly.

3. Create new view as a subclass of `dj_rest_auth.views.SocialLoginView` with `GitHubOAuth2Adapter` adapter, an `OAuth2Client` and a callback_url as attributes:

```
from allauth.socialaccount.providers.github.views import GitHubOAuth2Adapter
from allauth.socialaccount.providers.oauth2.client import OAuth2Client
from dj_rest_auth.registration.views import SocialLoginView

class GitHubLogin(SocialLoginView):
    adapter_class = GitHubOAuth2Adapter
    callback_url = CALLBACK_URL_YOU_SET_ON_GITHUB
    client_class = OAuth2Client
```

4. Create url for GitHubLogin view:

```
urlpatterns += [
    ...,
    path('dj-rest-auth/github/', GitHubLogin.as_view(), name='github_login')
]
```

Google

If you are using Google for your social authentication, you can choose `Authorization Code Grant` or `Implicit Grant` (deprecated). Serializer of dj-rest-auth accepts both `code` and `token`

1. Add `allauth.socialaccount` and `allauth.socialaccount.providers.google` apps to INSTALLED_APPS in your django settings.py:

```
INSTALLED_APPS = (
    ...,
    'rest_framework',
    'rest_framework.authtoken',
    'dj_rest_auth'
    ...,
    'django.contrib.sites',
    'allauth',
    'allauth.account',
    'dj_rest_auth.registration',
    ...,
    'allauth.socialaccount',
    'allauth.socialaccount.providers.google',

)
```

3. Create new view as a subclass of `dj_rest_auth.views.SocialLoginView` with `GoogleOAuth2Adapter` adapter, an `OAuth2Client` and a callback_url as attributes:

```
from allauth.socialaccount.providers.google.views import GoogleOAuth2Adapter
from allauth.socialaccount.providers.oauth2.client import OAuth2Client
from dj_rest_auth.registration.views import SocialLoginView

class GoogleLogin(SocialLoginView): # if you want to use Authorization Code Grant,␣
↪use this
    adapter_class = GoogleOAuth2Adapter
    callback_url = CALLBACK_URL_YOU_SET_ON_GOOGLE
    client_class = OAuth2Client

class GoogleLogin(SocialLoginView): # if you want to use Implicit Grant, use this
    adapter_class = GoogleOAuth2Adapter
```

4. Create url for GoogleLogin view:

```
urlpatterns += [
    ...,
    path('dj-rest-auth/google/', GoogleLogin.as_view(), name='google_login')
]
```

5. Retrive code (or token) By accessing Google's endpoint, you can get `code` or `token`

If you're using Authorization Code Grant, you can get code from following URL

```
https://accounts.google.com/o/oauth2/v2/auth?redirect_uri=<CALLBACK_URL_YOU_SET_ON_GOOGLE>&
CLIENT ID>&scope=openid%20email%20profile&access_type=offline
```

If you're using Implicit Grant, you can get token from following URL

```
https://accounts.google.com/o/oauth2/v2/auth?redirect_uri=<CALLBACK_URL_YOU_SET_ON_GOOGLE>&
CLIENT ID>&scope=openid%20email%20profile
```

6. POST code or token to specified URL(/dj-rest-auth/google/)

Additional Social Connect Views

If you want to allow connecting existing accounts in addition to login, you can use connect views:

```
from allauth.socialaccount.providers.facebook.views import FacebookOAuth2Adapter
from allauth.socialaccount.providers.github.views import GitHubOAuth2Adapter
from allauth.socialaccount.providers.twitter.views import TwitterOAuthAdapter
from allauth.socialaccount.providers.oauth2.client import OAuth2Client
from dj_rest_auth.registration.views import SocialConnectView
from dj_rest_auth.social_serializers import TwitterConnectSerializer

class FacebookConnect(SocialConnectView):
    adapter_class = FacebookOAuth2Adapter

class TwitterConnect(SocialConnectView):
    serializer_class = TwitterConnectSerializer
    adapter_class = TwitterOAuthAdapter

class GithubConnect(SocialConnectView):
    adapter_class = GitHubOAuth2Adapter
    callback_url = CALLBACK_URL_YOU_SET_ON_GITHUB
    client_class = OAuth2Client
```

In urls.py:

```
urlpatterns += [
    ...,
    path('dj-rest-auth/facebook/connect/', FacebookConnect.as_view(), name='fb_connect
↪')
    path('dj-rest-auth/twitter/connect/', TwitterConnect.as_view(), name='twitter_
↪connect')
    path('dj-rest-auth/github/connect/', GithubConnect.as_view(), name='github_connect
↪')
]
```

You can also use the following views to check all social accounts attached to the current authenticated user and disconnect selected social accounts:

```
from dj_rest_auth.registration.views import (
    SocialAccountListView, SocialAccountDisconnectView
)

urlpatterns += [
    ...,
    path(
        'socialaccounts/',
        SocialAccountListView.as_view(),
        name='social_account_list'
```

(continues on next page)

(continued from previous page)

```
    ),
    path(
        'socialaccounts/<int:pk>/disconnect/',
        SocialAccountDisconnectView.as_view(),
        name='social_account_disconnect'
    )
]
```

1.2.3 JSON Web Token (JWT) Support (optional)

By default `dj-rest-auth` uses Django's Token-based authentication. If you want to use JWT authentication, follow these steps:

1. **Install djangorestframework-simplejwt**
    - `djangorestframework-simplejwt` is currently the only supported JWT library.
2. Add a simple_jwt auth configuration to the list of authentication classes.

```
REST_FRAMEWORK = {
    ...
    'DEFAULT_AUTHENTICATION_CLASSES': (
        ...
        'dj_rest_auth.jwt_auth.JWTCookieAuthentication',
    )
    ...
}
```

3. Add the following configuration value to your settings file to enable JWT authentication in dj-rest-auth.

```
REST_AUTH = {
    ...
    'USE_JWT': True,
}
```

4. Declare what you want the cookie key to be called. If you want to use the refresh token feature, also be sure to set that variable.

```
REST_AUTH = {
    ...
    'JWT_AUTH_COOKIE': 'my-app-auth',
    'JWT_AUTH_REFRESH_COOKIE': 'my-refresh-token',
}
```

This example value above will cause dj-rest-auth to return a *Set-Cookie* header that looks like this:

```
Set-Cookie: my-app-auth=xxxxxxxxxxxxx; expires=Sat, 28 Mar 2020 18:59:00 GMT;␣
↪HttpOnly; Max-Age=300; Path=/
```

If `JWT_AUTH_REFRESH_COOKIE` is also set, it will also set a comparable cookie for that. `JWT_AUTH_COOKIE` is also used while authenticating each request against protected views.

1.3 API endpoints

1.3.1 Basic

- /dj-rest-auth/login/ (POST)
  - username
  - email
  - password

  Returns Token key
- /dj-rest-auth/logout/ (POST)

  Note: `ACCOUNT_LOGOUT_ON_GET = True` to allow logout using GET - this is the exact same configuration from allauth. NOT recommended, see: http://django-allauth.readthedocs.io/en/latest/views.html#logout
- /dj-rest-auth/password/reset/ (POST)
  - email
- /dj-rest-auth/password/reset/confirm/ (POST)
  - uid
  - token
  - new_password1
  - new_password2

  Note: uid and token are sent in email after calling /dj-rest-auth/password/reset/
- /dj-rest-auth/password/change/ (POST)
  - new_password1
  - new_password2
  - old_password

  Note: `OLD_PASSWORD_FIELD_ENABLED = True` to use old_password.

  Note: `LOGOUT_ON_PASSWORD_CHANGE = False` to keep the user logged in after password change
- /dj-rest-auth/user/ (GET, PUT, PATCH)
  - username
  - first_name

  - last_name

  Returns pk, username, email, first_name, last_name

- /dj-rest-auth/token/verify/ (POST)

  - token

  Returns an empty JSON object.

Note: `USE_JWT = True` to use token/verify/ route.

Note: Takes a token and indicates if it is valid. This view provides no information about a token's fitness for a particular use. Will return a `HTTP 200 OK` in case of a valid token and `HTTP 401 Unauthorized` with `{"detail": "Token is invalid or expired", "code": "token_not_valid"}` in case of a invalid or expired token.

- /dj-rest-auth/token/refresh/ (POST) (see also)

  - refresh

  Returns access

Note: `USE_JWT = True` to use token/refresh/ route.

Note: Takes a refresh type JSON web token and returns an access type JSON web token if the refresh token is valid. `HTTP 401 Unauthorized` with `{"detail": "Token is invalid or expired", "code": "token_not_valid"}` in case of a invalid or expired token.

1.3.2 Registration

- /dj-rest-auth/registration/ (POST)
  - username
  - password1
  - password2
  - email
- /dj-rest-auth/registration/verify-email/ (POST)

  - key

Note: If you set account email verification as mandatory, you have to add the VerifyEmailView with the used *name*. You need to import the view: `from dj_rest_auth.registration.views import VerifyEmailView`. Then add the url with the corresponding name: `path('dj-rest-auth/account-confirm-email/', VerifyEmailView.as_view(), name='account_email_verification_sent')` to the urlpatterns list.

- /dj-rest-auth/registration/resend-email/ (POST)

  - email

Resends the email verification

1.3.3 Social Media Authentication

Basing on example from installation section *Installation*

- /dj-rest-auth/facebook/ (POST)
  - access_token
  - code

> **Note:** `access_token` OR `code` can be used as standalone arguments, see https://github.com/iMerica/dj-rest-auth/blob/master/dj_rest_auth/registration/views.py

- /dj-rest-auth/twitter/ (POST)
  - access_token
  - token_secret

1.4 Configuration

dj-rest-auth behaviour can be controlled by adjust settings in `settings.py`:

```
# Django project settings.py

...

REST_AUTH = {
    'LOGIN_SERIALIZER': 'dj_rest_auth.serializers.LoginSerializer',
    'TOKEN_SERIALIZER': 'dj_rest_auth.serializers.TokenSerializer',
    'JWT_SERIALIZER': 'dj_rest_auth.serializers.JWTSerializer',
    'JWT_SERIALIZER_WITH_EXPIRATION': 'dj_rest_auth.serializers.
↪JWTSerializerWithExpiration',
    'JWT_TOKEN_CLAIMS_SERIALIZER': 'rest_framework_simplejwt.serializers.
↪TokenObtainPairSerializer',
    'USER_DETAILS_SERIALIZER': 'dj_rest_auth.serializers.UserDetailsSerializer',
    'PASSWORD_RESET_SERIALIZER': 'dj_rest_auth.serializers.PasswordResetSerializer',
    'PASSWORD_RESET_CONFIRM_SERIALIZER': 'dj_rest_auth.serializers.
↪PasswordResetConfirmSerializer',
    'PASSWORD_CHANGE_SERIALIZER': 'dj_rest_auth.serializers.PasswordChangeSerializer',

    'REGISTER_SERIALIZER': 'dj_rest_auth.registration.serializers.RegisterSerializer',

    'REGISTER_PERMISSION_CLASSES': ('rest_framework.permissions.AllowAny',),

    'TOKEN_MODEL': 'rest_framework.authtoken.models.Token',
    'TOKEN_CREATOR': 'dj_rest_auth.utils.default_create_token',

    'PASSWORD_RESET_USE_SITES_DOMAIN': False,
    'OLD_PASSWORD_FIELD_ENABLED': False,
    'LOGOUT_ON_PASSWORD_CHANGE': False,
```

(continues on next page)

(continued from previous page)

```
    'SESSION_LOGIN': True,
    'USE_JWT': False,

    'JWT_AUTH_COOKIE': None,
    'JWT_AUTH_REFRESH_COOKIE': None,
    'JWT_AUTH_REFRESH_COOKIE_PATH': '/',
    'JWT_AUTH_SECURE': False,
    'JWT_AUTH_HTTPONLY': True,
    'JWT_AUTH_SAMESITE': 'Lax',
    'JWT_AUTH_RETURN_EXPIRATION': False,
    'JWT_AUTH_COOKIE_USE_CSRF': False,
    'JWT_AUTH_COOKIE_ENFORCE_CSRF_ON_UNAUTHENTICATED': False,
}
```

Above, the default values for these settings are shown.

1.5 LOGIN_SERIALIZER

The path to the serializer class for the login in `dj_rest_auth.views.LoginView`. The value is the dotted path to `dj_rest_auth.serializers.LoginSerializer`, which is also the default.

1.6 TOKEN_SERIALIZER

The path to the serializer class for the Token response on successful authentication in `dj_rest_auth.views.LoginView`. The value is the dotted path to `dj_rest_auth.serializers.TokenSerializer`, which is also the default.

1.7 JWT_SERIALIZER

The path to the serializer class for the JWT response on successful authentication in `dj_rest_auth.views.LoginView`. The value is the dotted path to `dj_rest_auth.serializers.JWTSerializer`, which is also the default. Requires `USE_JWT=True` in settings.

1.8 JWT_SERIALIZER_WITH_EXPIRATION

The path to the serializer class for the JWT response with its expiration time on successful authentication in `dj_rest_auth.views.LoginView`. The value is the dotted path to `dj_rest_auth.serializers.JWTSerializerWithExpiration`, which is also the default. Requires `USE_JWT=True` in settings.

1.9 JWT_TOKEN_CLAIMS_SERIALIZER

The path to the serializer class for the JWT Claims on successful authentication in `dj_rest_auth.views.LoginView`. The value is the dotted path to `rest_framework_simplejwt.serializers.TokenObtainPairSerializer`, which is also the default. Requires `USE_JWT=True` in settings.

1.10 USER_DETAILS_SERIALIZER

The path to the serializer class for the User details in `dj_rest_auth.views.UserDetailsView`. The value is the dotted path to `dj_rest_auth.serializers.UserDetailsSerializer`, which is also the default.

1.11 PASSWORD_RESET_SERIALIZER

The path to the serializer class for the password reset in `dj_rest_auth.views.PasswordResetView`. The value is the dotted path to `dj_rest_auth.serializers.PasswordResetSerializer`, which is also the default.

1.12 PASSWORD_RESET_CONFIRM_SERIALIZER

The path to the serializer class for the password reset confirm in `dj_rest_auth.views.PasswordResetConfirmView`. The value is the dotted path to `dj_rest_auth.serializers.PasswordResetConfirmSerializer`, which is also the default.

1.13 PASSWORD_CHANGE_SERIALIZER

The path to the serializer class for the password change in `dj_rest_auth.views.PasswordChangeView`. The value is the dotted path to `dj_rest_auth.serializers.PasswordChangeSerializer`, which is also the default.

1.14 REGISTER_SERIALIZER

The path to the serializer class for the register in `dj_rest_auth.registration.views.RegisterView`. The value is the dotted path to `dj_rest_auth.registration.serializers.RegisterSerializer`, which is also the default.

Note: The custom `REGISTER_SERIALIZER` must define a `def save(self, request)` method that returns a user model instance.

1.15 REGISTER_PERMISSION_CLASSES

A tuple that contains paths to the permission classes for the register in `dj_rest_auth.registration.views.RegisterView`. The value is the dotted path to `path.to.another.permission.class`. `rest_framework.permissions.AllowAny` is included by default.

1.16 TOKEN_MODEL

The path to the model class for the token. The value is the dotted path to `rest_framework.authtoken.models.Token`, which is also the default. If set to `None` token authentication will be disabled. In this case at least one of `SESSION_LOGIN` or `USE_JWT` must be enabled.

1.17 TOKEN_CREATOR

The path to callable for creating tokens. The value is the dotted path to `dj_rest_auth.utils.default_create_token`, which is also the default.

1.18 PASSWORD_RESET_USE_SITES_DOMAIN

If set to `True`, the domain in the password reset e-mail will be set to the domain you defined in `django.contrib.sites` module with `SITE_ID=1`. Default is `False`.

1.19 OLD_PASSWORD_FIELD_ENABLED

If set to `True`, old password verification in `dj_rest_auth.views.PasswordChangeView` will be added. Default is `False`.

1.20 LOGOUT_ON_PASSWORD_CHANGE

If set to `True`, current user will be logged out after a password change. Default is `False`.

1.21 SESSION_LOGIN

If set to `True`, session login in `dj_rest_auth.views.LoginView` will be enabled. Default is `True`.

1.22 USE_JWT

If set to `True`, JWT Authentication in `dj_rest_auth.views.LoginView` will be used instead of Token or Session based login. Default is `False`.

Note: JWT Authentication in dj-rest-auth is built on top of djangorestframework-simplejwt https://github.com/SimpleJWT/django-rest-framework-simplejwt. You must install it in order to be able to use JWT Authentication in dj-rest-auth.

1.23 JWT_AUTH_COOKIE

The cookie name for `access_token` from JWT Authentication. Default is `None`.

1.24 JWT_AUTH_REFRESH_COOKIE

The cookie name for `refresh_token` from JWT Authentication. Default is `None`.

1.25 JWT_AUTH_REFRESH_COOKIE_PATH

The cookie path for `refresh_token` from JWT Authentication. Default is `'/'`.

1.26 JWT_AUTH_SECURE

If set to `True`, the cookie will only be sent through https scheme. Default is `False`.

1.27 JWT_AUTH_HTTPONLY

If set to `True`, the client-side JavaScript will not be able to access the cookie. Default is `True`.

Note: `refresh_token` will not be sent if `JWT_AUTH_HTTPONLY` set to `True`, set it to `False` if you need `refresh_token`.

1.28 JWT_AUTH_SAMESITE

To tell the browser not to send this cookie when performing a cross-origin request. Default is `'Lax'`. SameSite isn't supported by all browsers.

1.29 JWT_AUTH_RETURN_EXPIRATION

If set to `True`, the `access_token` and `refresh_token` expiration time will be included in response on successful JWT Authentication in `dj_rest_auth.views.LoginView`. Default is `False`.

1.30 JWT_AUTH_COOKIE_USE_CSRF

If set to `True`, enable CSRF checks for only authenticated views when using the JWT cookie for auth. Does not effect a client's ability to authenticate using a JWT Bearer Auth header without a CSRF token. Default is `False`.

1.31 JWT_AUTH_COOKIE_ENFORCE_CSRF_ON_UNAUTHENTICATED

If set to `True`, enables CSRF checks for authenticated and unauthenticated views when using the JWT cookie for auth. It does not effect a client's ability to authenticate using a JWT Bearer Auth header without a CSRF token (though getting the JWT token in the first place without passing a CSRF token isnt possible). Default is `False`.

1.32 Demo project

This demo project shows how you can potentially use dj-rest-auth app with jQuery on frontend. To run this locally follow the steps below.

```
cd /tmp
git clone https://github.com/iMerica/dj-rest-auth.git
cd dj-rest-auth/demo/
pip install -r requirements.pip
python manage.py migrate --settings=demo.settings --noinput
python manage.py runserver --settings=demo.settings
```

Now, go to `http://127.0.0.1:8000/` in your browser. There is also a Single Page Application (SPA) in React within the `demo/` directory. To run this do:

```
cd react-spa/
yarn # or npm install
yarn run start
```

Now, go to `https://localhost:3000` in your browser to view it.

1.33 FAQ

1. Why account_confirm_email url is defined but it is not usable?

   In /dj_rest_auth/registration/urls.py we can find something like this:

   ```
   url(r'^account-confirm-email/(?P<key>[-:\w]+)/$', TemplateView.as_view(),
       name='account_confirm_email'),
   ```

   This url is used by django-allauth. Empty TemplateView is defined just to allow reverse() call inside app - when email with verification link is being sent.

   You should override this view/url to handle it in your API client somehow and then, send post to /verify-email/ endpoint with proper key. If you don't want to use API on that step, then just use ConfirmEmailView view from: django-allauth https://github.com/pennersr/django-allauth/blob/master/allauth/account/views.py

2. I get an error: Reverse for 'password_reset_confirm' not found.

   You need to add *password_reset_confirm* url into your `urls.py` (at the top of any other included urls). Please check the `urls.py` module inside demo app example for more details.

3. How can I update UserProfile assigned to User model?

   Assuming you already have UserProfile model defined like this

```
from django.db import models
from django.contrib.auth.models import User

class UserProfile(models.Model):
    user = models.OneToOneField(User, related_name='userprofile')
    # custom fields for user
    company_name = models.CharField(max_length=100)
```

To allow update user details within one request send to dj_rest_auth.views.UserDetailsView view, create serializer like this:

```
from rest_framework import serializers
from dj_rest_auth.serializers import UserDetailsSerializer

class UserProfileSerializer(serializers.ModelSerializer):
    class Meta:
        model = UserProfile
        fields = ('company_name',)

class UserSerializer(UserDetailsSerializer):

    profile = UserProfileSerializer(source="userprofile")

    class Meta(UserDetailsSerializer.Meta):
        fields = UserDetailsSerializer.Meta.fields + ('profile',)

    def update(self, instance, validated_data):
        userprofile_serializer = self.fields['profile']
        userprofile_instance = instance.userprofile
        userprofile_data = validated_data.pop('userprofile', {})

        # to access the 'company_name' field in here
        # company_name = userprofile_data.get('company_name')

        # update the userprofile fields
        userprofile_serializer.update(userprofile_instance, userprofile_
↪data)

        instance = super().update(instance, validated_data)
        return instance
```

And setup USER_DETAILS_SERIALIZER in django settings:

```
REST_AUTH = {
    ...
    'USER_DETAILS_SERIALIZER': 'demo.serializers.UserSerializer',
}
```

1.34 Vulnerability Disclosure Policy

Please observe the standard best practices of responsible disclosure, especially considering that this is OSS. See OWASP's disclosure cheat sheet.

Some basic rules:

- Keep it legal.

- Respect everyone's privacy.
- Contact the core maintainer(s) immediately if you discover a serious security vulnerability (imichael@pm.me for now).