# django-rest-auth Documentation

***Release 0.3.0***

**Tivix Inc.**

November 24, 2015

**Warning:** Updating django-rest-auth to version **0.3.4** is highly recommended because of a security issue in PasswordResetConfirmation validation method.

**Note:** django-rest-auth from v0.3.3 supports django-rest-framework v3.0

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 django-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

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

### 1.1.3 Angular app

- Tivix has also created angular module which uses API endpoints from this app - angular-django-registration-auth

### 1.1.4 Demo project

- You can also check our Demo Project which is using jQuery on frontend.

## 1.2 Installation

1. Install package:

```
pip install django-rest-auth
```

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

```
INSTALLED_APPS = (
    ...,
    'rest_framework',
    'rest_framework.authtoken',
    ...,
    '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 rest_auth urls:

```
urlpatterns = patterns('',
    ...,
    url(r'^rest-auth/', include('rest_auth.urls'))
)
```

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` - see this doc for installation http://django-allauth.readthedocs.org/en/latest/installation.html.
2. Add `allauth`, `allauth.account` and `rest_auth.registration` apps to INSTALLED_APPS in your django settings.py:

```
INSTALLED_APPS = (
    ...,
    'allauth',
    'allauth.account',
    'rest_auth.registration',
)
```

3. Add rest_auth.registration urls:

```
urlpatterns = patterns('',
    ...,
    (r'^rest-auth/', include('rest_auth.urls')),
    (r'^rest-auth/registration/', include('rest_auth.registration.urls'))
)
```

### 1.2.2 Social Authentication (optional)

Using `django-allauth`, `django-rest-auth` provides helpful class for creating social media authentication view. Below is an example with Facebook authentication.

**Note:** Points 1, 2 and 3 are related with `django-allauth` configuration, so if you have already configured social authentication, then please go to step 4. See `django-allauth` documentation for more details.

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

```
INSTALLED_APPS = (
    ...,
    'rest_framework',
    'rest_framework.authtoken',
    'rest_auth'
    ...,
    'allauth',
    'allauth.account',
    'rest_auth.registration',
    ...,
    'allauth.socialaccount',
    'allauth.socialaccount.providers.facebook',
)
```

2. Add Social Application in django admin panel

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

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

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

4. Create url for FacebookLogin view:

```
urlpatterns += pattern('',
    ...,
    url(r'^rest-auth/facebook/$', FacebookLogin.as_view(), name='fb_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.

## 1.3 API endpoints

### 1.3.1 Basic

- /rest-auth/login/ (POST)
  - username (string)
  - password (string)
- /rest-auth/logout/ (POST)
- /rest-auth/password/reset/ (POST)
  - email
- /rest-auth/password/reset/confirm/ (POST)

   - uid
   - token
   - new_password1
   - new_password2

**Note:** uid and token are sent in email after calling /rest-auth/password/reset/

- /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

- /rest-auth/user/ (GET)
- /rest-auth/user/ (PUT/PATCH)
   - username
   - first_name
   - last_name
   - email

### 1.3.2 Registration

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

**Note:** This endpoint is based on `allauth.account.views.SignupView` and uses the same form as in this view. To override fields you have to create custom Signup Form and define it in django settings:

```
ACCOUNT_FORMS = {
    'signup': 'path.to.custom.SignupForm'
}
```

See allauth documentation for more details.

- /rest-auth/registration/verify-email/ (POST)
   - key

### 1.3.3 Social Media Authentication

Basing on example from installation section Installation

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

## 1.4 Configuration

- **REST_AUTH_SERIALIZERS**

  You can define your custom serializers for each endpoint without overriding urls and views by adding `REST_AUTH_SERIALIZERS` dictionary in your django settings. Possible key values:

  - LOGIN_SERIALIZER - serializer class in `rest_auth.views.LoginView`, default value `rest_auth.serializers.LoginSerializer`
  - TOKEN_SERIALIZER - response for successful authentication in `rest_auth.views.LoginView`, default value `rest_auth.serializers.TokenSerializer`
  - USER_DETAILS_SERIALIZER - serializer class in `rest_auth.views.UserDetailsView`, default value `rest_auth.serializers.UserDetailsSerializer`
  - PASSWORD_RESET_SERIALIZER - serializer class in `rest_auth.views.PasswordResetView`, default value `rest_auth.serializers.PasswordResetSerializer`
  - PASSWORD_RESET_CONFIRM_SERIALIZER - serializer class in `rest_auth.views.PasswordResetConfirmView`, default value `rest_auth.serializers.PasswordResetConfirmSerializer`
  - PASSWORD_CHANGE_SERIALIZER - serializer class in `rest_auth.views.PasswordChangeView`, default value `rest_auth.serializers.PasswordChangeSerializer`

  Example configuration:

```
REST_AUTH_SERIALIZERS = {
    'LOGIN_SERIALIZER': 'path.to.custom.LoginSerializer',
    'TOKEN_SERIALIZER': 'path.to.custom.TokenSerializer',
    ...
}
```

- **REST_SESSION_LOGIN** - Enable session login in Login API view (default: True)
- **OLD_PASSWORD_FIELD_ENABLED** - set it to True if you want to have old password verification on password change enpoint (default: False)
- **LOGOUT_ON_PASSWORD_CHANGE** - set to False if you want to keep the current user logged in after a password change

## 1.5 Demo project

The idea of creating demo project was to show how you can potentially use django-rest-auth app with jQuery on frontend. Do these steps to make it running (ideally in virtualenv).

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

Now, go to `http://127.0.0.1:8000/` in your browser.

## 1.6 FAQ

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

   In /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: djang-allauth https://github.com/pennersr/django-allauth/blob/master/allauth/account/views.py#L190

2. 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)
    # custom fields for user
    company_name = models.CharField(max_length=100)
```

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

```
from rest_framework import serializers
from rest_auth.serializers import UserDetailsSerializer

class UserSerializer(UserDetailsSerializer):

    company_name = serializers.CharField(source="userprofile.company_name")

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

    def update(self, instance, validated_data):
        profile_data = validated_data.pop('userprofile', {})
        company_name = profile_data.get('company_name')

        instance = super(UserSerializer, self).update(instance, validated_data)

        # get and update user profile
        profile = instance.userprofile
        if profile_data and company_name:
            profile.company_name = company_name
            profile.save()
        return instance
```

And setup USER_DETAILS_SERIALIZER in django settings:

```
REST_AUTH_SERIALIZERS = {
    'USER_DETAILS_SERIALIZER': 'demo.serializers.UserSerializer'
}
```

## 1.7 Changelog

### 1.7.1 0.6.0

- dropped support for Python 2.6
- dropped support for Django 1.6
- fixed demo code
- added better validation support for serializers
- added optional logout after password change
- compatibility fixes
- bugfixes

### 1.7.2 0.5.0

- replaced request.DATA with request.data for compatibility with DRF 3.2
- authorization codes for social login
- view classes rename (appended "View" to all of them)
- bugfixes

### 1.7.3 0.4.0

- Django 1.8 compatiblity fixes

### 1.7.4 0.3.4

- fixed bug in PasswordResetConfirmation serializer (token field wasn't validated)
- fixed bug in Register view

### 1.7.5 0.3.3

- support django-rest-framework v3.0

### 1.7.6 0.3.2

- fixed few minor bugs

### 1.7.7 0.3.1

- added old_password field in PasswordChangeSerializer
- make all endpoints browsable
- removed LoggedInRESTAPIView, LoggedOutRESTAPIView
- fixed minor bugs

### 1.7.8 0.3.0

- replaced `django-registration` with `django-allauth`
- moved registration logic to separated django application (`rest_auth.registration`)
- added serializers customization in django settings
- added social media authentication view
- changed request method from GET to POST in logout endpoint
- changed request method from POST to PUT/PATCH for user details edition
- changed password reset confim url - uid and token should be sent in POST
- increase test coverage
- made compatibile with django 1.7
- removed user profile support