# Contents

1 Contents
  1.1 Introduction ......................................................... 3
  1.2 Installation .......................................................... 4
  1.3 API endpoints .......................................................... 8
  1.4 Configuration .......................................................... 10
  1.5 Demo project ........................................................... 11
  1.6 FAQ ................................................................. 11
  1.7 Changelog .............................................................. 12
Warning: Updating django-rest-auth from version 0.3.3 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
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:

   ```python
   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:

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

4. Migrate your database

   ```bash
   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 django-rest-auth[with_social].

2. Add django.contrib.sites, allauth, allauth.account and rest_auth.registration apps to INSTALLED_APPS in your django settings.py:

3. Add SITE_ID = 1 to your django settings.py
SITE_ID = 1

3. Add rest_auth.registration urls:

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

**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:

```python
INSTALLED_APPS = (
    ...
    'rest_framework',
    'rest_framework.authtoken',
    'rest_auth',
    ...
    'django.contrib.sites',
    'allauth',
    'allauth.account',
    '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 `rest_auth.registration.views.SocialLoginView` with FacebookOAuth2Adapter adapter as an attribute:

```python
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:

```python
class FacebookLogin:
    adapter_class = FacebookOAuth2Adapter
    serializer_class = FacebookLoginSerializer

urlpatterns += [
    url(r'^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 `rest_auth.registration.views.SocialLoginView` with `TwitterOAuthAdapter` adapter and `TwitterLoginSerializer` as an attribute:

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

urlpatterns += [
    url(r'^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](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 `rest_auth.views.SocialLoginView` with `GitHubOAuth2Adapter` adapter, an `OAuth2Client` and a `callback_url` as attributes:

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

urlpatterns += [
    url(r'^rest-auth/github/$', GithubLogin.as_view(), name='github_login')
]
```
Additional Social Connect Views

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

```python
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 rest_auth.registration.views import SocialConnectView
from 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:

```python
urlpatterns += [
    ...,
    url(r'^rest-auth/facebook/connect/$', FacebookConnect.as_view(), name='fb_connect'),
    url(r'^rest-auth/twitter/connect/$', TwitterConnect.as_view(), name='twitter_connect'),
    url(r'^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:

```python
from rest_auth.registration.views import (SocialAccountListView, SocialAccountDisconnectView)

urlpatterns += [
    ...,
    url(r'^socialaccounts/$', SocialAccountListView.as_view(), name='social_account_list' ),
    url(r'^socialaccounts/(?P<pk>[d]+)/disconnect/$', SocialAccountDisconnectView.as_view(), name='social_account_disconnect' )
]
```
1.2.3 JWT Support (optional)

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

1. Install djangorestframework-jwt
   • djangorestframework-jwt is currently the only supported JWT library.

2. The JWT_PAYLOAD_HANDLER and JWT_ENCODE_HANDLER settings are imported from the djangorestframework object.
   • Refer to the library’s documentation for information on using different encoders.

3. Add the following configuration value to your settings file to enable JWT authentication.

```
REST_USE_JWT = True
```

1.3 API endpoints

1.3.1 Basic

• /rest-auth/login/ (POST)
  – username
  – email
  – password
  Returns Token key

• /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

• /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, PUT, PATCH)
  - username
  - first_name
  - last_name
  Returns pk, username, email, first_name, last_name

1.3.2 Registration

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

Note: access_token OR code can be used as standalone arguments, see https://github.com/Tivix/django-rest-auth/blob/master/rest_auth/registration/views.py

- /rest-auth/twitter/ (POST)
  - access_token
  - token_secret
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
  - JWT_SERIALIZER - (Using REST_USE_JWT=True) response for successful authentication in rest_auth.views.LoginView, default value rest_auth.serializers.JWTSerializer
  - 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:

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

• **REST_AUTH_REGISTER_SERIALIZERS**

  You can define your custom serializers for registration endpoint. Possible key values:

  - REGISTER_SERIALIZER - serializer class in rest_auth.registration.views.RegisterView, default value rest_auth.registration.serializers.RegisterSerializer

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

• **REST_AUTH_TOKEN_MODEL** - model class for tokens, default value rest_framework.authtoken.models

• **REST_AUTH_TOKEN_CREATOR** - callable to create tokens, default value rest_auth.utils.default_create_token.

• **REST_SESSION_LOGIN** - Enable session login in Login API view (default: True)
• REST_USE_JWT - Enable JWT Authentication instead of Token/Session based. This is built on top of django-rest-framework-jwt http://getblimp.github.io/django-rest-framework-jwt/, which must also be installed. (default: False)

• OLD_PASSWORD_FIELD_ENABLED - set it to True if you want to have old password verification on password change endpoint (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).

```bash
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 migrate --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:

   ```python
   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

   ```python
   from django.db import models
   from django.contrib.auth.models import User
   
   class UserProfile(models.Model):
       user = models.OneToOneField(User)
   ```

(continues on next page)
To allow update user details within one request send to rest_auth.views.UserDetailsView view, create serializer like this:

```python
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:

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

### 1.7 Changelog

#### 1.7.1 0.9.5

- fixed package distribution issue

#### 1.7.2 0.9.4

- Compatibility fixes (#437, #506)
- JWT auth cookie fix (#345)
- config & packaging fixes
- updated docs
- added new translations (Czech, Chinese, Turkish, Korean)
1.7.3 0.9.3

- added social connect views
- added check for pre-existing accounts in social login
- prevent double-validation in LoginSerializer
- unit tests and demo project changes for Django 2.0

1.7.4 0.9.2

- added permission classes configuration for registration
- added more info to JWT docs
- added Polish translations

1.7.5 0.9.1

- fixed import error when extending rest_auth serializers
- added sensitive fields decorator
- added Spanish translations

1.7.6 0.9.0

- allowed using custom UserDetailsSerializer with JWTSerializer
- fixed error with logout on GET
- updated api endpoints and configuration docs
- bugfixes
- minor text fixes

1.7.7 0.8.2

- fixed allauth import error
- added swagger docs to demo project

1.7.8 0.8.1

- added support for django-allauth hmac email confirmation pattern

1.7.9 0.8.0

- added support for django-rest-framework-jwt
- bugfixes
1.7.10 0.7.0

- Wrapped API returned strings in ugettext_lazy
- Fixed not using get_username which caused issues when using custom user model without username field
- Django 1.9 support
- Added TwitterLoginSerializer

1.7.11 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

1.7.12 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

1.7.13 0.4.0

- Django 1.8 compatibility fixes

1.7.14 0.3.4

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

1.7.15 0.3.3

- support django-rest-framework v3.0

1.7.16 0.3.2

- fixed few minor bugs
1.7.17 0.3.1

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

1.7.18 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 confirm url - uid and token should be sent in POST
- increase test coverage
- made compatiibile with django 1.7
- removed user profile support