django-rest-auth Documentation

lordpeara

Aug 25, 2019

Contents

django-rest-framework-auth is a authentication provider for django and rest_framework.

With very simple instructions, you can add your authentication API.

CHAPTER 1

Quickstart

Just install it, including urls and see APIs from your browsable API.

```
$ pip install django-rest-framework-auth
$ django-admin startproject proj
$ vi proj/proj/settings.py
```

```
# settings.py
# ...
INSTALLED_APPS = (
    # ...
    'rest_auth',
    'rest_auth.users',
    'rest_framework',
    # ...
)

# urls.py
# ...
urlpatterns += [
    url(r'^auth/', include(('rest_auth.urls'))),
    url(r'^auth/user/', include(('rest_auth.users.urls'))),
]
```

```
$ python manage.py runserver
```

see API lists! http://localhost:8000/auth/api-root/

CHAPTER 2

Contents

2.1 Installation

Install package

```
$ pip install django-rest-framework-auth
```

Add `rest_framework` to `INSTALLED_APPS` in settings.py

```
INSTALLED_APPS = (
    # ...
    'rest_auth',
    'rest_framework',
    # required by 3 apps, auth, contenttypes and sessions.
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',

    # NOTE place `rest_auth` upper than `django.contrib.admin` if
    # you wanted to adopt email templates `rest_auth` have.
    # (or you see admin's templates)
    # You can ignore that if you write your own email template.
    # (also you should place your own app upper.)
    'django.contrib.admin',
    # And also you should add `django.contrib.staticfiles` to see
    # rest_framework's templates from HTMLRenderers
    'django.contrib.staticfiles',
    # ...
)
```

Add `rest_auth.urls` to your `urls.py`

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

(continues on next page)

(continued from previous page)

```
    url(r'^auth/user/', include(('rest_auth.users.urls'))),
]
```

2.2 rest_auth API reference

Django Rest Framework Auth provides very simple & quick way to adopt authentication APIs' to your django project.

2.2.1 Rationale

django-rest-framework's *Serializer* is nice idea for detaching business logic from view functions. It's very similar to django's `Form`, but serializer is not obligible for rendering response data, and should not. - django forms also do this, seriously!!! some expert beginners just know form is ONLY FOR *html form rendering* :(

Unluckily, even though django already provides forms and views for authentication, We cannot use these for REST-APIs. It uses forms!! (rest_framework does not use forms.)

We think there should be some serializers & views (or viewsets) to use `rest_framework`'s full features. (such as throttling, pagination, versioning or content-negotiations)

Let's have a good taste of these elegant implementations.

2.2.2 API Endpoints

Below API endpoints can be re-configured if you write your urls.py

- **POST /login/**
  - username
  - password

  authenticate user and persist him/her to website
- **POST /logout/** let a user logged out.

Note: Logout from HTTP GET is not implemented.

- **POST /forgot/**
  - email

  send a link for resetting password to user
- **GET /reset/{uid64}/{token}/**
  - uid64, token - automatically generated tokens (when email is sent)
  - new_password
  - new_password (confirm)

  reset a password for user
- **GET /reset/d/** a view seen by user after resetting password
- **POST /change-password/**

  - old_password
  - new_password
  - new_password (confirm)

  change a password for user

- **GET /api-root/**
  - see api lists

2.2.3 API Endpoints (`rest_auth.users`)

Below API endpoints can be accessed if you add `rest_auth.users` into `INSTALLED_APPS` and add `rest_auth.users.urls` into your `urls.py`

- **POST /user/**
  - username
  - email
  - password
  - confirm_password

  Create a user.

  verification e-mail is sent when you set `REST_AUTH_SIGNUP_REQUIRE_EMAIL_CONFIRMATION`

- GET /user/

  Show user list. Only privacy-safe user fields are visible.

- GET /user/{pk}

  Show a user.

- DELETE /user/{pk}

  Delete a user.

- GET /user/v/{uid64}/{token}/

  Verify user. After verification, user can use full features of websites.

2.2.4 Index

rest_auth.serializers

Serializer implementations for authentication.

class `rest_auth.serializers.`**LoginSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Serializer for loggin in. It checks username and password are correct for settings.AUTH_USER_MODEL.

After validating it, `user` instance created for authenticated user. View methods should persist this user. (through `django.contrib.auth.login`)

Parameters

- **username** – `USERNAME_FIELD` for AUTH_USER_MODEL
- **password** – user's password

validate(*data*)
Checks username & password. uses `django.contrib.auth.authenticate`

Parameters **data** – validated data from `Serializer.validate`

Returns validated_data

Raises **VaildationError** – if username or password are incorrect

confirm_login_allowed(*user*)
Checks if validated user is allowed for website.

Override this method if you use custom authentication method and have additional methods for allowing login.

Raises **VaildationError** – if user are not allowed

create(*validated_data*)
persist a authenticated user in this step.

Parameters **validated_data** – validated_data should contains `request`. You should pass request to serialzer.save.

perform_login(*request*, *user*)
Persist a user. Override this method if you do more than persisting user.

get_user()

Returns `user` instance created after `self.validate`

class rest_auth.serializers.**PasswordResetSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)
Sends a website link for resetting password. It uses django's `PasswordResetForm` directly because there is just one required field, *email*, and form implemented its business logic nicely.

Parameters **email** – email address to receive password-reset-link.

password_reset_form_class
alias of `django.contrib.auth.forms.PasswordResetForm`

validate_email(*value*)

Raises

- **VaildationError** – `rest_framework`'s field validation
- **VaildationError** – `django`'s field vaildation

save(*domain_override=None*, *subject_template_name='registration/password_reset_subject.txt'*, *email_template_name='registration/password_reset_email.html'*, *use_https=True*, *token_generator=<django.contrib.auth.tokens.PasswordResetTokenGenerator object>*, *from_email=None*, *request=None*, *html_email_template_name=None*, *extra_email_context=None*)
sends a email, which contains link for resetting password

class rest_auth.serializers.**SetPasswordSerializer**(*user*, **args*, ***kwargs*)
This serializer resets password of a given user. Please be VERY CAREFUL for using this any given user's password can be changed.

Setting permission IsAdminUser is recommended.

Parameters

- **new_password1** – new password

- **new_password2** – new password confirmation.

validate(*data*)

Raises **VaildationError** – if two given passwords are different.

create(*validated_data*)
resets password

class rest_auth.serializers.**PasswordChangeSerializer**(*user*, **args*, ***kwargs*)
resets password of user. Resetting password is done if old_password is correct and two new passwords are equals.

Parameters

- **old_password** – old_password
- **new_password1** – new password
- **new_password2** – new password confirmation.

validate_old_password(*old_password*)

Raises **ValidationError** – if old_password is not correct

rest_auth.views

Views for authentication

In this views, authentication views are composited with GenericAPIView (of rest_framework) and mixins, which is implemented for process views.

Because, we didn't know what methods you use for process business logics. You can construct your own views by extending our mixins.

(rest_framework's generic views used this strategy)

class rest_auth.views.**LoginMixin**
Mixin for logging-in

response_includes_data = False
Set this to `True` if you wanna send user data (or more) when authentication is successful. (default: `False`)

serializer_class = None
You should make your own serializer class if you cusomize auth backend and this backend are not satisfied by `LoginSerializer`.

(accept other than `username` and `password`. (e.g `RemoteUserBackend`)

login(*request*, **args*, ***kwargs*)
Main business logic for loggin in

Raises **ValidationError** – auth failed, but it will be handled by rest_frameworks error handler.

get_response_data(*data*)
Override this method when you use `response_includes_data` and You wanna send customized user data (beyond serializer.data)

class rest_auth.views.**LoginView**(***kwargs*)
LoginView for REST-API.

post(*request*, **args*, ***kwargs*)
Just calls `LoginMixin.login`

class rest_auth.views.**LogoutView**(***kwargs*)
LogoutView for user logout.

post(*request*, **args*, ***kwargs*)
Logout a user. performed by `django.contrib.auth.logout`

No data is to sent.

class rest_auth.views.**PasswordForgotMixin**
View for sending password-reset-link.

serializer_class
alias of *rest_auth.serializers.PasswordResetSerializer*

forgot(*request*, **args*, ***kwargs*)
Sends a password-reset-link to requested email.

class rest_auth.views.**PasswordForgotView**(***kwargs*)
sending password-reset email to user.

post(*request*, **args*, ***kwargs*)

class rest_auth.views.**PasswordForgotConfirmView**(***kwargs*)
django-rest-auth's password reset confirmation just adopts django's one. This idea is under assumption, which password reset confirmation should be done, by clicking password-reset-url we sent and moving to webpage to change password.

class rest_auth.views.**PasswordResetDoneView**(***kwargs*)
adopts django's password reset complete view.

class rest_auth.views.**PasswordChangeMixin**
Change password for a user.

serializer_class
alias of *rest_auth.serializers.PasswordChangeSerializer*

reset(*request*, **args*, ***kwargs*)
Reset password. No data is to sent.

class rest_auth.views.**PasswordChangeView**(***kwargs*)
View for change password.

post(*request*, **args*, ***kwargs*)

rest_auth API reference

`rest_auth.users` extra application

`rest_auth.users` Application is for user-related APIs.

You need not to use this app if you want to your own serializer and user views.

rest_auth.users.serializers

class rest_auth.users.serializers.**UserSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

User serializer for rest_framework & AUTH_USER_MODEL.

Fields & methods are built on a django's defualt `User` model. Extend this serializer if you need your custom user model.

(Even if `AUTH_USER_MODEL` is can be customized, this is recommended that You don't change & use customized user model. using custom user model is very complex.)

Parameters

- **username** – `USERNAME_FIELD` of `AUTH_USER_MODEL`
- **email** – `User.get_email_field_name()`
- **password1** – password of a user (write_only, used only when created)
- **password2** – password confirmation (write_only)

TODO Serializer Only implements creating. list/get are need to be implmtd

validate(*data*)

Vaildates if two passwords are equal.

Raises **ValidationError** – when 2 passwds are different

create(*validated_data*)

Creates user instance

CAVEAT:

A clear difference between django's `ModelForm` and rest_framework's `ModelSerializer` is that, model serializer's `save` method doesn't respect form's `commit=True`.

Inside `super().create`, a query is fired to create user, and inside this, additional query is fired to save hashed password. It's because `ModelSerializer`'s `create` method uses default manager's create function, `Model._default_manager.create()`

(User model creation is recommended by calling `UserManager`'s `create_user` method)

Parameters **validated_data** – validated data created after `self.vaildate`

send_mail(*user*, *domain_override=None*, *subject_template_name='registration/verify_email.txt'*, *email_template_name='registration/verify_email.html'*, *use_https=False*, *token_generator=<django.contrib.auth.tokens.PasswordResetTokenGenerator object>*, *from_email=None*, *request=None*, *html_email_template_name=None*, *extra_email_context=None*)

Send verification mail to newbie.

rest_auth.users.views

class rest_auth.users.views.**UserViewSet**(***kwargs*)

Viewset for UserModel.

serializer_class

alias of *rest_auth.users.serializers.UserSerializer*

class rest_auth.users.views.**EmailVerificationConfirmView**(***kwargs*)
Email verification view for newly-created User instances.

After user verified his/her email, users can use his/her full features of website.

rest_auth.contrib

Batteries included

There are utilized or patched functions for building ours.

rest_auth.contrib.rest_framework

rest_auth.contrib.rest_framework.decorators.**sensitive_post_parameters**(**parameters*)
hide sensitive POST paramters from Django's error reporting.

This decorator should be used for `rest_framework`'s views if your views use sensitive data like *password*, because rest_framework use `rest_framework.request.Request`, **NOT** `django.http.HttpRequest` (*This is not subclassed*)

(so django's `sensitive_post_parameters` cannot be used for rest_framework)

2.3 Configurations

Settings for rest_auth.

Settings used by rest_auth can be overriden from your *settings.py* file.

rest_auth.default_settings.**REST_AUTH_EMAIL_OPTIONS = {}**
Default: `{}`

Options for email, which is sent to reset password. Detail options guide **'here. <>'_**

rest_auth.default_settings.**REST_AUTH_LOGIN_EMPTY_RESPONSE = True**
Default: `True`

Set this to `False` if your LoginView should return non-empty response.

rest_auth.default_settings.**REST_AUTH_LOGIN_SERIALIZER_CLASS = 'rest_auth.serializers.LoginS**
Default: `"rest_auth.serializers.LoginSerializer"`

Serializer to log in. Update this if you use customized auth backend.

rest_auth.default_settings.**REST_AUTH_SIGNUP_REQUIRE_EMAIL_CONFIRMATION = False**
Default: `False`

If your sign-up process has verification-via-email, set this flag to `True` to send email.

> **Warning:** This functionality is not implemented yet.

2.4 Tricks and Tips

Python Module Index

r

Index

S

U

V