edX Django REST Framework Extensions Documentation

Release 1.2.0

edX

November 03, 2016

Contents

1 Requirements **3**

2 Installation **5**

3 Table of Contents **7**
- 3.1 Settings . . . 7
- 3.2 Authentication . . . 7
- 3.3 Permissions . . . 9
- 3.4 Utility Functions . . . 9

Python Module Index **11**

This package provides extensions for Django REST Framework that are useful for developing on the edX platform.

CHAPTER 1

Requirements

- Python (2.7, 3.5)
- Django (1.8, 1.9)
- Django REST Framework (3.2+)

CHAPTER 2

Installation

Install using `pip`:

```
$ pip install edx-drf-extensions
```

CHAPTER 3

Table of Contents

3.1 Settings

All settings for this package reside in a dict, *EDX_DRF_EXTENSIONS*. Within this dict, the following keys should be specified, depending on the functionality you are using.

3.1.1 BearerAuthentication

These settings are used by the *BearerAuthentication* class.

OAUTH2_USER_INFO_URL

Default: `None`

URL of an endpoint on the OAuth2 provider where *BearerAuthentication* can retrieve details about the user associated with the provided access token. This endpoint should return a JSON object with user details and `HTTP 200` if, and only if, the access token is valid. See *BearerAuthentication.process_user_info_response()* for an example of the expected data format.

3.1.2 JwtAuthentication

These settings are used by the *JwtAuthentication* class.

JWT_PAYLOAD_USER_ATTRIBUTES

Default: `('email',)`

The list of user attributes in the JWT payload that *JwtAuthentication* will use to update the local `User` model. These payload attributes should exactly match the names the attributes on the local `User` model.

3.2 Authentication

Authentication classes are used to associate a request with a user. Unless otherwise noted, all of the classes below adhere to the Django REST Framework's API for authentication classes.

class `BearerAuthentication`
Simple token based authentication.

This authentication class is useful for authenticating an OAuth2 access token against a remote authentication provider. Clients should authenticate by passing the token key in the "Authorization" HTTP header, prepended with the string *"Bearer "*.

This class relies on the OAUTH2_USER_INFO_URL being set to the value of an endpoint on the OAuth provider, that returns a JSON object with information about the user. See `process_user_info_response` for the expected format of this object. This data will be used to get, or create, a `User`. Additionally, it is assumed that a successful response from this endpoint (authenticated with the provided access token) implies the access token is valid.

Example Header: Authorization: Bearer 401f7ac837da42b97f613d789819ff93537bee6a

`authenticate_credentials`(*token*)
Validate the bearer token against the OAuth provider.

Parameters `token` (*str*) – Access token to validate

Returns

tuple containing:

user (User): User associated with the access token access_token (str): Access token

Return type (tuple)

Raises `AuthenticationFailed` – The user is inactive, or retrieval of user info failed.

`get_user_info`(*token*)
Retrieves the user info from the OAuth provider.

Parameters `token` (*str*) – OAuth2 access token.

Returns dict

Raises `UserInfoRetrievalFailed` – Retrieval of user info from the remote server failed.

`get_user_info_url`()
Returns the URL, hosted by the OAuth2 provider, from which user information can be pulled.

`process_user_info_response`(*response*)
Process the user info response data.

By default, this simply maps the edX user info key-values (example below) to Django-friendly names. If your provider returns different fields, you should sub-class this class and override this method.

```
{
    "username": "jdoe",
    "email": "jdoe@example.com",
    "first_name": "Jane",
    "last_name": "Doe"
}
```

Parameters `response` (*dict*) – User info data

Returns dict

class `JwtAuthentication`
JSON Web Token based authentication.

This authentication class is useful for authenticating a JWT using a secret key. Clients should authenticate by passing the token key in the "Authorization" HTTP header, prepended with the string *"JWT "*.

This class relies on the JWT_AUTH being configured for the application as well as JWT_PAYLOAD_USER_ATTRIBUTES being configured in the EDX_DRF_EXTENSIONS config.

At a minimum, the JWT payload must contain a username. If an email address is provided in the payload, it will be used to update the retrieved user's email address associated with that username.

Example Header: Authorization: JWT eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJmYzJiNzIwMTE0YmIwN2I
ImFkbWluaXN0cmF0b3IiOmZhbHNlLCJuYW1lIjoiaG9ub3IiLCJleHA.QHDXdo8gDJ5p9uOErTLZtl2HK_61kgLs71VHp
JUZpIYMkEd38uf1vj-4HZkzeNBnZZZ3Kdvq7F8ZioREPKNyEVSm2mnzl1v49EthehN9kwfUgFgPXfUh-
pCvLDqwCCTdAXMcTJ8qufzEPTYYY54lY

authenticate_credentials(*payload*)
Get or create an active user with the username contained in the payload.

get_jwt_claim_attribute_map()
Returns a mapping of JWT claims to user model attributes.

Returns dict

3.3 Permissions

Permissions determine whether a request should be granted or denied access. Unless otherwise noted, all of the classes below adhere to the Django REST Framework's API for permission classes.

class IsSuperuser
Allows access only to superusers.

3.4 Utility Functions

This module contains useful utility functions.

jwt_decode_handler(*token*)
Decodes a JSON Web Token (JWT).

Notes

- Requires "exp" and "iat" claims to be present in the token's payload.
- Supports multiple issuer decoding via settings.JWT_AUTH['JWT_ISSUERS'] (see below)
- Aids debugging by logging DecodeError and InvalidTokenError log entries when decoding fails.

Examples

Use with djangorestframework-jwt, by changing your Django settings:

```
JWT_AUTH = {
    'JWT_DECODE_HANDLER': 'edx_rest_framework_extensions.utils.jwt_decode_handler',
    'JWT_ISSUER': 'https://the.jwt.issuer',
    'JWT_SECRET_KEY': 'the-jwt-secret-key',  (defaults to settings.SECRET_KEY)
    'JWT_AUDIENCE': 'the-jwt-audience',
}
```

Enable multi-issuer support by specifying a list of dictionaries as settings.JWT_AUTH['JWT_ISSUERS']:

```
    JWT_ISSUERS = [
            {
                'ISSUER': 'test-issuer-1',
                'SECRET_KEY': 'test-secret-key-1',
                'AUDIENCE': 'test-audience-1',
            },
            {
                'ISSUER': 'test-issuer-2',
                'SECRET_KEY': 'test-secret-key-2',
                'AUDIENCE': 'test-audience-2',
            }
        ]
```

Parameters **token** (*str*) – JWT to be decoded.

Returns Decoded JWT payload.

Return type dict

Raises

- `MissingRequiredClaimError` – Either the exp or iat claims is missing from the JWT payload.
- `InvalidTokenError` – Decoding fails.

e

edx_rest_framework_extensions.authentication, 7
edx_rest_framework_extensions.permissions, 9
edx_rest_framework_extensions.utils, 9

A

authenticate_credentials() (BearerAuthentication method), 8
authenticate_credentials() (JwtAuthentication method), 9

B

BearerAuthentication (class in edx_rest_framework_extensions.authentication), 7

E

edx_rest_framework_extensions.authentication (module), 7
edx_rest_framework_extensions.permissions (module), 9
edx_rest_framework_extensions.utils (module), 9

G

get_jwt_claim_attribute_map() (JwtAuthentication method), 9
get_user_info() (BearerAuthentication method), 8
get_user_info_url() (BearerAuthentication method), 8

I

IsSuperuser (class in edx_rest_framework_extensions.permissions), 9

J

jwt_decode_handler() (in module edx_rest_framework_extensions.utils), 9
JwtAuthentication (class in edx_rest_framework_extensions.authentication), 8

P

process_user_info_response() (BearerAuthentication method), 8