

---

# **django-trench Documentation**

*Release 0.2.2*

**Merixstudio**

**May 21, 2019**



---

# Contents

---

<b>1</b>	<b>Contents</b>	<b>1</b>
1.1	About . . . . .	1
1.2	Installation . . . . .	3
1.3	Additional settings . . . . .	4
1.4	API Endpoints . . . . .	6
1.5	Authentication backends . . . . .	7
1.6	Examples . . . . .	9
<b>2</b>	<b>Indices and tables</b>	<b>13</b>



## 1.1 About

**django-trench** provides a set of REST API endpoints to supplement [django-rest-framework](#) with multi-factor authentication (MFA, 2FA). It supports both standard built-in authentication methods, as well as JWT (JSON Web Token). **django-trench** follows the url pattern developed in [djoser](#) library and may act as its supplement.

We deliver a couple of sample secondary authentication methods including sending OTP based code by email, SMS/text as well as through 3rd party mobile apps or utilising YubiKey. Developers can easily add own auth backend supporting any communication channel.

### 1.1.1 Features

- Easily pluggable and compatible with [django-rest-framework](#) and [djoser](#)
- Allows user to pick an additional authentication method from range of backends defined by a developer. Read more: [backends](#)
- Comes out of a box with email, SMS, mobile apps and YubiKey support

### 1.1.2 Requirements

#### Supported versions

- Python 3.4, 3.5, 3.6, 3.7
- Django 1.11, 2.0, 2.1, 2.2
- Django REST Framework 3.7, 3.8, 3.9

If you implement Token Based Authentication:

- `djoser >= 1.21.0`

If you are going to use JWT authentication:

- `django-rest-framework-jwt >= 1.11.0`

or

- `django-rest-framework-simplejwt >= 3.3`

### 1.1.3 Quick Start

1. Install the package using pip:

```
pip install django-trench
```

or add it to your requirements file.

2. Add `trench` library to `INSTALLED_APPS` in your app settings file:

```
INSTALLED_APPS = (  
    ...  
    'rest_framework',  
    'rest_framework.authtoken', # In case of implementing Token Based Authentication  
    ...  
    'trench',  
)
```

3. Run migrations

Read further in: [installation](#).

### 1.1.4 Translation

Trench uses Transifex service to translate our package into other languages.

We will appreciate your help with translation.

<https://www.transifex.com/merixstudio/django-trench/dashboard/>

### 1.1.5 Demo project

You can also check our [live demo](#).

## 1.2 Installation

### 1.2.1 First steps

1. Install the package using pip:

```
pip install django-trench
```

or add it to your requirements file.

2. Add `trench` library to `INSTALLED_APPS` in your app settings file:

```
INSTALLED_APPS = (
    ...,
    'rest_framework',
    'rest_framework.authtoken', # In case of implementing Token Based Authentication
    ...,
    'trench',
)
```

**Note:** If you're going to use `djoser` to handle user authentication make sure you have it installed and included in `INSTALLED_APPS`. You'll also need `django-rest-framework-jwt` to support JSON Web Tokens.

### 1.2.2 Config

#### `urls.py`

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

If you utilise `djoser` and JWT authentication:

```
urlpatterns = [
    ...,
    url(r'^auth/', include('trench.urls')), # Base endpoints
    url(r'^auth/', include('djoser.urls')),
    url(r'^auth/', include('trench.urls.djoser')), # for Token Based Authorization
    url(r'^auth/', include('trench.urls.jwt')), # for django-rest-framework-jwt
    url(r'^auth/', include('trench.urls.simplejwt')), # for django-rest-framework-
    ↪ simplejwt
]
```

#### `settings.py`

`django-trench` supports `django-rest-framework` built-in Token Based Authentication, as well as JSON Web Tokens. You'll need setup it accordingly:

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework.authentication.TokenAuthentication',
        # or / and
        'rest_framework_jwt.authentication.JSONWebTokenAuthentication',
        # or / and
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ),
}
```

### 1.2.3 Migrations

Last but not least, run migrations:

```
$ ./manage.py migrate
```

## 1.3 Additional settings

You can customize settings by adding TRENCH\_AUTH dict in your settings.py:

```
TRENCH_AUTH = {
    'FROM_EMAIL': 'your@email.com',
    'USER_ACTIVE_FIELD': 'is_active',
    'BACKUP_CODES_QUANTITY': 5,
    'BACKUP_CODES_LENGTH': 10, # keep (quantity * length) under 200
    'BACKUP_CODES_CHARACTERS': (
        'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
    ),
    'ENCRYPT_BACKUP_CODES': True,
    'SECRET_KEY_LENGTH': 16,
    'DEFAULT_VALIDITY_PERIOD': 30,
    'CONFIRM_DISABLE_WITH_CODE': False,
    'CONFIRM_BACKUP_CODES_REGENERATION_WITH_CODE': True,
    'ALLOW_BACKUP_CODES_REGENERATION': True,
    'APPLICATION_ISSUER_NAME': 'MyApplication',
    'MFA_METHODS': {
        'email': {
            'VERBOSE_NAME': _('email'),
            'VALIDITY_PERIOD': 60 * 10,
            'FIELD': 'email',
            'HANDLER': 'trench.backends.basic_mail.SendMailBackend',
            'SERIALIZER': 'trench.serializers.RequestMFACreateEmailSerializer',
            'SOURCE_FIELD': 'email',
        },
        ...,
    },
}
```



### 1.3.1 FROM\_EMAIL

Email address to be used as sender's while using email backend for sending codes.

### 1.3.2 USER\_ACTIVE\_FIELD

Field on `User` model which stores information whether user's account is active or not. Default: `is_active`

### 1.3.3 BACKUP\_CODES\_QUANTITY

Number of backup codes to be generated.

### 1.3.4 BACKUP\_CODES\_LENGTH

Length of backup code.

### 1.3.5 BACKUP\_CODES\_CHARACTERS

Range of characters to be used in backup code.

### 1.3.6 ENCRYPT\_BACKUP\_CODES

Backup codes to be encrypted before saving. Default: `True`

### 1.3.7 SECRET\_KEY\_LENGTH

Length of the shared secret key. For compatibility with Google Authenticator minimum is 8 (16 on Android) and to a power of 2. <https://github.com/antonioribeiro/google2fa#google-authenticator-secret-key-compatibility> Default: 16

### 1.3.8 DEFAULT\_VALIDITY\_PERIOD

Period when OTP code validates positively (in seconds). Becomes a default if no validity period has been declared on a specific authentication method.

### 1.3.9 CONFIRM\_DISABLE\_WITH\_CODE

If `True` requires a code verification to disable a current authentication method. Default: `False`

### 1.3.10 CONFIRM\_BACKUP\_CODES\_REGENERATION\_WITH\_CODE

If `True` requires a code verification to regenerate backup code.

### 1.3.11 ALLOW\_BACKUP\_CODES\_REGENERATION

If `True` allows regenerate backup codes. Default: `True`

### 1.3.12 APPLICATION\_ISSUER\_NAME

Issuer name for QR generation.

### 1.3.13 MFA\_METHODS

A dictionary which holds all authentication methods and its settings. New method can be added as a next item.

#### Method item properties

- 'VERBOSE\_NAME' method name
- 'VALIDITY\_PERIOD' OTP code validity
- 'HANDLER' location of the method's handler
- 'SERIALIZER' location of a serializer
- 'SOURCE\_FIELD' field on a User model utilised in the method (i.e. field storing phone number for SMS)

## 1.4 API Endpoints

### 1.4.1 MFA method activation

- `/[method name]/activate/ [POST]`

Request a new method activation and get an authentication code by specified channel.

Payload:

- method MFA method name

- `/[method name]/activate/confirm/` [POST]`

Accepts the auth code, activates the method and returns backup codes

Payload:

- code auth code received by specified channel

- `/[method name]/deactivate/` [POST]`

Deactivates the specified method. Depending on *Additional settings* sends out a auth code and requires confirmation.

Payload:

- code auth code received by specified channel

[method\_name] one of MFA methods specified in your project `settings.py`. Check out *Additional settings*.

- `/code/request/ [POST]`

Triggers sending out a code.

## 1.4.2 Login

- `/login/` [POST]

First step, if 2FA is enable returns `ephemeral_token` required in next step as well as current auth method, otherwise logs in user.

Payload: `* username * password`

- `/login/code/` [POST]

Requires token generated in previous step and OTP code, logs in user (returns token)

Payload: `* ephemeral_token * code`

## 1.4.3 Backup codes

- `/mfa/codes/regenerate/` [POST]

Requests new batch of backup codes.

Payload:

- `method` MFA method name

## 1.4.4 Settings

- `/mfa/config/` [GET]

Display app's configuration

- `/mfa/user-active-methods/` [GET]

Display methods activated by user

- `/mfa/change-primary-method/` [POST]

Change default authentication method

Payload:

- `method` MFA method name
- `code` auth code received by specified channel

## 1.5 Authentication backends

`django-trench` comes with three predefined authentication methods.

Custom backends can be easily added by inheriting `AbstractMessageDispatcher` class.

### 1.5.1 Built-in backends

#### Email

This basic method utilise build-in Django backend. You'll need to have Email Backend setup. Check out [Django documentation](#).

### Text/SMS

SMS backends sends out text messages with Twilio or Smsapi.pl. Credentials can be set in method's specific settings.

```
TRENCH_AUTH = {
    (...)
    'MFA_METHODS': {
        'sms_twilio': {
            'VERBOSE_NAME': 'sms',
            'VALIDITY_PERIOD': 60 * 10,
            'HANDLER': 'trench.backends.twilio.TwilioBackend',
            'SOURCE_FIELD': 'phone_number',
            'TWILIO_ACCOUNT_SID': TWILIO SID,
            'TWILIO_AUTH_TOKEN': TWILIO TOKEN,
            'TWILIO_VERIFIED_FROM_NUMBER': TWILIO REGISTERED NUMBER,
        },
        ...,
    },
}
```

Read more in *Additional settings*.

### Authentication apps

This backend returns OTP based QR link to be scanned by apps like Google Authenticator and Authy.

```
TRENCH_AUTH = {
    (...)
    'MFA_METHODS': {
        'app': {
            'VERBOSE_NAME': 'app',
            'VALIDITY_PERIOD': 60 * 10,
            'USES_THIRD_PARTY_CLIENT': True,
            'HANDLER': 'trench.backends.application.ApplicationBackend',
        },
        ...,
    },
}
```

### YubiKey

```
TRENCH_AUTH = {
    (...)
    'MFA_METHODS': {
        'yubi': {
            'VERBOSE_NAME': 'yubi',
            'HANDLER': 'trench.backends.yubikey.YubiKeyBackend',
            'SOURCE_FIELD': 'yubikey_id',
            'YUBICLOUD_CLIENT_ID': '',
        },
        ...,
    },
}
```

(continues on next page)

(continued from previous page)

```
    },
}
```

## 1.5.2 Adding own authentication method

Base on provided examples you can create own handler class, which heritates from `AbstractMessageDispatcher`.

```
from trench.backends import AbstractMessageDispatcher

class CustomAuthBackend(AbstractMessageDispatcher):

    def dispatch_message(self, *args, **kwargs):
        (...)
        return {'data': 'ok'}
```

It may be also required to provide a custom serializer depending on what information need to be passed on from user. In order to run your own method update settings as follows:

```
TRENCH_AUTH = {
    (...)
    'MFA_METHODS': {
        'yourmethod': {
            'VERBOSE_NAME': 'yourmethod',
            'VALIDITY_PERIOD': 60 * 10,
            'SOURCE_FIELD': 'phone_number', # if your backend requires custom field_
↪ on User model
            'HANDLER': 'yourapp.backends.CustomAuthBackend',
            'SERIALIZER': 'yourapp.serializers.CustomAuthSerializer',
        },
        ...,
    },
}
```

## 1.6 Examples

In order to let you familiarise with the library, a fully working test project is provided in the repository. It allows you to run `django-trench` with basic settings as well as play with it thanks to a sample frontend app.

### 1.6.1 Launching a sample app

1. Clone the repository:

```
$ git clone https://github.com/merixstudio/django-trench.git
```

2. Check `testproject` directory and adjust `settings.py` inside `testapp` according to *Installation* and *Additional settings* if necessary.
3. Make sure you have `docker` and `docker-compose` installed. Use `Makefile` to run backend:

```
$ make build
$ make migrate
```

3. Run the app using command:

```
$ make client
```

Frontend app is available on <http://localhost:3000/> and expects backend running on <http://localhost:8000/>

### 1.6.2 Basic usage

You can create an admin user to be able to access admin panel <http://localhost:8000/admin>:

```
$ make create_admin
```

From built-in admin panel you can add users and setup credentials.

Alternatively `djoser` endpoints can be used to manage users in through REST requests. Read further in [djoser docs](#).

Let's login:

```
$ curl -X POST http://localhost:8000/auth/login/ -d 'username=admin&
↳password=yourpassword'
```

In the following request you'll need a provided `token` for authorization.

To activate an email authentication:

```
$ curl -X POST http://localhost:8000/auth/email/activate/ -d 'method=email'
-H 'Authorization: JWT [token provided]'
```

Check the code and confirm:

```
$ curl -X POST http://localhost:8000/auth/email/activate/confirm/ -d 'code=[code_
↳provided]'
-H 'Authorization: JWT [token provided]'
```

In response you'll receive a batch of backup codes.

Let's login again and check if an extra authentication works.

```
$ curl -X POST http://localhost:8000/auth/login/ -d 'username=admin&
↳password=yourpassword'

{
  "ephemeral_token": "token",
  "method": "email",
  "other_methods": []
}
```

Right, the code has been dispatched by the primary method.

Now we only need pass on the code and ephemeral\_token:

```
$ curl -X POST http://localhost:8000/auth/login/code/
-d 'code=[code from previous step]&ephemeral_token=[ephemeral_token from step before]'

{
  "token": "JWT token",
}
```

All right, we're in!





## CHAPTER 2

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`