

---

# **drf-user**

***Release 1.1.0***

**Jan 03, 2023**



---

## Contents

---

<b>1</b>	<b>Overview</b>	<b>3</b>
<b>2</b>	<b>Feature List</b>	<b>5</b>
<b>3</b>	<b>Contents</b>	<b>7</b>
3.1	Installation . . . . .	7
3.1.1	Getting the code . . . . .	7
3.1.2	Requirements . . . . .	7
3.1.3	Prerequisites . . . . .	7
3.1.4	Manual Settings . . . . .	9
3.2	API Documentation . . . . .	10
3.2.1	Register . . . . .	10
3.2.2	Login . . . . .	10
3.2.3	Account . . . . .	11
3.2.4	OTP . . . . .	12
3.2.5	OTP Register Login . . . . .	13
3.2.6	Reset Password . . . . .	13
3.2.7	Is Unique . . . . .	14
3.2.8	Upload Image . . . . .	14
3.2.9	Refresh Token . . . . .	14
3.3	Indices and tables . . . . .	15
	<b>HTTP Routing Table</b>	<b>17</b>



User APP for Django REST Framework with API Views.

---

**Note:** One of the winners of [PyCharm Project of the Decade Competition](#)

---



# CHAPTER 1

---

## Overview

---

Django REST Framework - User is a Django app that overrides default user app to provide additional attributes and functionalities.





## CHAPTER 2

---

### Feature List

---

JWT Support (Using [Simple JWT](#))

Mobile Number

Single field for full name

REST API to register

REST API to login

MultiModelBackend: User can login using either of mobile, email or username

REST API to login with OTP (Same API endpoint as for OTP Verification; Set `is_login: true` while sending JSON request)

OTP Verification for mobile and email

API to register / login with OTP (no pre-registration required)

API to set user's profile image

Mail sending feature upon successful registration

SMS sending feature upon successful registration

Change Password

Update Profile

Generic Configuration based on *settings.py*

Signal based mails: Pending in OTP section

Mail based activation (optional alternative for OTP based activation)

Social Auth Endpoints(Login using fb/google)



### 3.1 Installation

Each of the following steps needs to be configured for the *drf-user* to be fully functional.

#### 3.1.1 Getting the code

Install from PyPI (recommended) with `pip`:

```
pip install drf_user
```

Or Install via `easy_install`:

```
easy_install drf_user
```

Or Install from source code:

```
pip install -e git+https://github.com/101Loop/drf-user#egg=drf_user
```

#### 3.1.2 Requirements

`drf-user` supports Python 3.7 and above.

#### 3.1.3 Prerequisites

- Add `drf_user` and other dependencies in `INSTALLED_APPS` of your projects `settings.py`

```
INSTALLED_APPS = [
    ...
    'drf_user',
    'rest_framework',
    'django_filters',
    ...
]
```

- Include urls of *drf\_user* in your projects `urls.py`

```
from django.urls import path

urlpatterns = [
    ...
    path('api/user/', include('drf_user.urls')),
    ...
]
```

Or if you have *regex* based urls use *re\_path*

```
from django.urls import re_path

urlpatterns = [
    ...
    re_path(r'^api/user/', include('drf_user.urls')),
    ...
]
```

- Include `AUTH_USER_MODEL` in `settings.py`

```
...
AUTH_USER_MODEL = 'drf_user.User'
...
```

- Set `AUTHENTICATION_BACKEND` in `settings.py`

```
AUTHENTICATION_BACKENDS = [
    'drf_user.auth.MultiFieldModelBackend', # to support login with email/mobile
]
```

- Set `DEFAULT_AUTHENTICATION_CLASSES` in `REST_FRAMEWORK` configuration in your `settings.py`

```
REST_FRAMEWORK = {
    ...
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
        ...
    ),
    ...
}
```

- Set `SIMPLE_JWT` configurations in `settings.py` (*these are default values from Simple JWT, update as per your requirements*)

```
from datetime import timedelta

...
```

(continues on next page)

(continued from previous page)

```
# see https://django-rest-framework-simplejwt.readthedocs.io/en/latest/settings.html
SIMPLE_JWT = {
    "ACCESS_TOKEN_LIFETIME": timedelta(minutes=5),
    "REFRESH_TOKEN_LIFETIME": timedelta(days=1),
    "ROTATE_REFRESH_TOKENS": False,
    "BLACKLIST_AFTER_ROTATION": True,
    "UPDATE_LAST_LOGIN": True,
    "ALGORITHM": "HS256",
    "SIGNING_KEY": SECRET_KEY,
    "VERIFYING_KEY": None,
    "AUDIENCE": None,
    "ISSUER": None,
    "AUTH_HEADER_TYPES": ("Bearer",),
    "AUTH_HEADER_NAME": "HTTP_AUTHORIZATION",
    "USER_ID_FIELD": "id",
    "USER_ID_CLAIM": "user_id",
    "AUTH_TOKEN_CLASSES": ("rest_framework_simplejwt.tokens.AccessToken",),
    "TOKEN_TYPE_CLAIM": "token_type",
    "JTI_CLAIM": "jti",
    "SLIDING_TOKEN_REFRESH_EXP_CLAIM": "refresh_exp",
    "SLIDING_TOKEN_LIFETIME": timedelta(minutes=5),
    "SLIDING_TOKEN_REFRESH_LIFETIME": timedelta(days=1),
}
```

- Finally, run migrate command

```
python manage.py migrate drf_user
```

### 3.1.4 Manual Settings

User can define manual user config in `settings.py` file in `USER_SETTINGS` variable. Default options are provided below, update as per your requirements.

```
USER_SETTINGS = {
    "MOBILE_OPTIONAL": True,
    'DEFAULT_ACTIVE_STATE': False,
    'OTP': {
        'LENGTH': 7,
        'ALLOWED_CHARS': '1234567890',
        'VALIDATION_ATTEMPTS': 3,
        'SUBJECT': 'OTP for Verification',
        'COOLING_PERIOD': 3
    },
    'MOBILE_VALIDATION': True,
    'EMAIL_VALIDATION': True,
    'REGISTRATION': {
        'SEND_MAIL': False,
        'SEND_MESSAGE': False,
        'MAIL_SUBJECT': 'Welcome to DRF-USER',
        'SMS_BODY': 'Your account has been created',
        'TEXT_MAIL_BODY': 'Your account has been created.',
        'HTML_MAIL_BODY': 'Your account has been created.'
    }
}
```

## 3.2 API Documentation

### 3.2.1 Register

API Docs for Register.

**POST /register/**

Register a new user to the system.

```
{
  "username": "username",
  "name": "name",
  "email": "email@user.com",
  "mobile": "9999999999",
  "password": "password"
}
```

#### JSON Parameters

- **username** (*str*) – unique username
- **name** (*str*) – name of the user
- **email** (*str*) – unique email of user
- **mobile** (*str*) – unique mobile number of user
- **password** (*str*) – password of user

#### Status Codes

- 201 [Created](#) – if supplied params are valid
- 400 [Bad Request](#) – if supplied params are invalid

### 3.2.2 Login

API Docs for Login.

**POST /login/**

Login a user to the system.

```
{
  "username": "username",
  "password": "my_secret_password",
}
```

#### JSON Parameters

- **username** (*str*) – unique username
- **password** (*str*) – password of user

#### Status Codes

- 200 [OK](#) – if supplied params are valid
- 400 [Bad Request](#) – if some fields are missing
- 401 [Unauthorized](#) – if supplied params are invalid

### 3.2.3 Account

API Docs for Account.

**GET** `/account/`  
Get a user.

```
{
  "id": 1,
  "username": "dummy_username",
  "name": "dummy_name",
  "email": "email@dummy.com",
  "mobile": "9999999999",
  "is_superuser": true,
  "is_staff": true
}
```

#### Status Codes

- 200 OK – if request is authenticated
- 401 Unauthorized – if request is not authenticated

**PUT** `/account/`  
Update all details of user.

```
{
  "username": "updated_username",
  "name": "updated_name",
  "email": "email@updated.com",
  "mobile": "9999999999",
  "password": "updated_password"
}
```

#### JSON Parameters

- **username** (*str*) – unique username
- **name** (*str*) – name of the user
- **email** (*str*) – unique email of user
- **mobile** (*str*) – unique mobile number of user
- **password** (*str*) – password of user

#### Status Codes

- 200 OK – if request is authenticated
- 400 Bad Request – if any param is not supplied
- 401 Unauthorized – if request is not authenticated

**PATCH /account/**

Update some details of user.

```
{
  "name": "partial_updated_name",
  "email": "email@partial_updated.com",
}
```

**JSON Parameters**

- **username** (*str*) – unique username, optional
- **name** (*str*) – name of the user, optional
- **email** (*str*) – unique email of user, optional
- **mobile** (*str*) – unique mobile number of user, optional
- **password** (*str*) – password of user, optional

**Status Codes**

- 200 OK – if request is authenticated
- 400 Bad Request – if any param is not supplied
- 401 Unauthorized – if request is not authenticated

### 3.2.4 OTP

API Docs for OTP.

**POST /otp/**

Generate, validate and login using OTP.

```
{
  "destination": "1234567890",
  "email": "email@django.com",
  "verify_otp": "123456",
  "is_login": "True",
  "_comment1": "destination can be email/mobile",
  "_comment2": "when using mobile as destination, use email",
  "_comment3": "to verify otp, add verify_otp to request",
  "_comment4": "for log in, just add is_login to request",
}
```

**JSON Parameters**

- **destination** (*str*) – destination where otp to be sent
- **email** (*str*) – if mobile is used in destination then use this for email, optional
- **verify\_otp** (*str*) – to verify otp, optional
- **is\_login** (*str*) – to login user, optional

**Status Codes**

- 201 Created – if supplied params are valid
- 400 Bad Request – if supplied params are invalid
- 403 Forbidden – if supplied otp is invalid



### 3.2.5 OTP Register Login

API Docs for OTP Register Login.

**POST** `/otpreglogin/`

Register, Login using OTP.

```
{
  "name": "some_awesome_name",
  "email": "email@django.com",
  "mobile": "1234567890",
  "verify_otp": "123456",
}
```

#### JSON Parameters

- **name** (*str*) – name of user
- **email** (*str*) – email of user
- **mobile** (*str*) – mobile of user
- **verify\_otp** (*str*) – to verify otp, optional

#### Status Codes

- **201 Created** – if supplied params are valid
- **400 Bad Request** – if supplied params are invalid
- **403 Forbidden** – if supplied otp is invalid

### 3.2.6 Reset Password

API Docs for Reset Password.

**POST** `/password/reset/`

Reset user's password.

- To reset user's password, first you have to call `/otp/` with `is_login` parameter value false.
- Then call this API

```
{
  "email": "email@django.com",
  "otp": "123456",
  "password": "my_new_secret_password",
}
```

#### JSON Parameters

- **email** (*str*) – email of user
- **otp** (*str*) – otp received on email
- **password** (*str*) – new password

#### Status Codes

- **202 Accepted** – if supplied params are valid
- **400 Bad Request** – if supplied params are invalid

- 403 Forbidden – if supplied otp is invalid

### 3.2.7 Is Unique

API Docs for Is Unique.

#### POST /isunique/

Check uniqueness of username, email, mobile.

```
{
  "prop": "email",
  "value": "email@django.com"
}
```

#### JSON Parameters

- **prop** (*str*) – property to check for uniqueness, choices are username, email, mobile
- **value** (*str*) – value to check for uniqueness

#### Status Codes

- 200 OK – if supplied params are valid
- 400 Bad Request – if supplied params are invalid

### 3.2.8 Upload Image

API Docs for Upload Image.

#### POST /upload-image/

Upload user's profile image.

```
{
  "profile_image": "<file_name>"
}
```

#### JSON Parameters

- **profile\_image** (*file*) – image file

#### Status Codes

- 201 Created – if supplied params are valid
- 400 Bad Request – if image not passed
- 401 Unauthorized – if supplied token is invalid

### 3.2.9 Refresh Token

API Docs for Refresh Token.

#### POST /refresh-token/

When short-lived access token expires, you can use the longer-lived refresh token to obtain another access token.

```
{  
  "refresh": "generated refresh token"  
}
```

#### JSON Parameters

- **refresh** (*str*) – refresh token

#### Status Codes

- 200 OK – if supplied refresh token is valid
- 400 Bad Request – if refresh token is not passed
- 401 Unauthorized – if refresh token is invalid

## 3.3 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)



---

## HTTP Routing Table

---

### /account

GET /account/, 11  
PUT /account/, 11  
PATCH /account/, 12

### /isunique

POST /isunique/, 14

### /login

POST /login/, 10

### /otp

POST /otp/, 12

### /otpreglogin

POST /otpreglogin/, 13

### /password

POST /password/reset/, 13

### /refresh-token

POST /refresh-token/, 14

### /register

POST /register/, 10

### /upload-image

POST /upload-image/, 14