
djoser Documentation

Release 1.1.5

Piotr Szpetkowski

Aug 29, 2019

1	Introduction	1
2	Getting started	3
2.1	Available endpoints	3
2.2	Supported authentication backends	3
2.3	Supported Python versions	4
2.4	Supported Django versions	4
2.5	Supported Django Rest Framework versions	4
2.6	Installation	4
2.7	Configuration	4
3	Sample usage	7
4	Authentication Backends	9
4.1	Token Based Authentication	9
4.2	JSON Web Token Authentication	10
5	Settings	11
5.1	PASSWORD_RESET_CONFIRM_URL	11
5.2	SEND_ACTIVATION_EMAIL	11
5.3	SEND_CONFIRMATION_EMAIL	12
5.4	ACTIVATION_URL	12
5.5	SET_USERNAME_RETYPE	12
5.6	SET_PASSWORD_RETYPE	12
5.7	PASSWORD_RESET_CONFIRM_RETYPE	12
5.8	LOGOUT_ON_PASSWORD_CHANGE	12
5.9	USER_EMAIL_FIELD_NAME	12
5.10	PASSWORD_RESET_SHOW_EMAIL_NOT_FOUND	13
5.11	TOKEN_MODEL	13
5.12	SERIALIZERS	13
5.13	EMAIL	14
5.14	SOCIAL_AUTH_TOKEN_STRATEGY	14
5.15	SOCIAL_AUTH_ALLOWED_REDIRECT_URIS	14
5.16	PERMISSIONS	14
6	Base Endpoints	17
6.1	User	17

6.2	User Create	18
6.3	User Delete	18
6.4	User Activate	18
6.5	User Resend Activation E-mail	19
6.6	Set Username	19
6.7	Set Password	19
6.8	Reset Password	20
6.9	Reset Password Confirmation	20
7	Token Endpoints	21
7.1	Token Create	21
7.2	Token Destroy	21
8	JWT Endpoints	23
8.1	JWT Create	23
8.2	JWT Refresh	23
8.3	JWT Verify	23
9	Social Endpoints	25
9.1	Provider Auth	25
10	Signals	27
10.1	user_registered	27
10.2	user_activated	27
11	Migration Guide	29
11.1	Migrating from 1.3 to 1.4	29
11.2	Migrating from 1.1 to 1.2	29
11.3	Migrating from 0.x to 1.0	29
12	Emails	31
13	Adjustment	33
14	Examples	35
14.1	Early detecting invalid password reset tokens	35
15	Indices and tables	37

CHAPTER 1

Introduction

REST implementation of [Django](#) authentication system. **djoser** library provides a set of [Django Rest Framework](#) views to handle basic actions such as registration, login, logout, password reset and account activation. It works with [custom user model](#).

Instead of reusing Django code (e.g. `PasswordResetForm`), we reimplemented few things to fit better into [Single Page App](#) architecture.

Developed by [SUNSCRAPERS](#) with passion & patience.

2.1 Available endpoints

- `/users/`
- `/users/me/`
- `/users/confirm/`
- `/users/change_username/`
- `/password/`
- `/password/reset/`
- `/password/reset/confirm/`
- `/token/login/` (Token Based Authentication)
- `/token/logout/` (Token Based Authentication)
- `/jwt/create/` (JSON Web Token Authentication)
- `/jwt/refresh/` (JSON Web Token Authentication)
- `/jwt/verify/` (JSON Web Token Authentication)

2.2 Supported authentication backends

- Token based authentication from [DRF](#)
- JSON Web Token authentication from [django-rest-framework-simplejwt](#)

2.3 Supported Python versions

- Python 2.7
- Python 3.4
- Python 3.5
- Python 3.6

2.4 Supported Django versions

- Django 1.11
- Django 2.0

2.5 Supported Django Rest Framework versions

- Django Rest Framework 3.7

2.6 Installation

```
$ pip install -U djoser
```

If you are going to use JWT authentication, you will also need to install `django-rest-framework-simplejwt` with:

```
$ pip install -U django-rest-framework-simplejwt
```

Finally if you are going to use third party based authentication e.g. facebook, you will need to install `social-auth-app-django` with:

```
$ pip install -U social-auth-app-django
```

2.7 Configuration

Configure `INSTALLED_APPS`:

```
INSTALLED_APPS = (  
    'django.contrib.auth',  
    (...),  
    'rest_framework',  
    'djoser',  
    (...),  
)
```

Configure `urls.py`:


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

HTTP Basic Auth strategy is assumed by default as Django Rest Framework does it. We strongly discourage and do not provide any explicit support for basic auth. You should customize your authentication backend as described in *Authentication Backends*.

In case of third party based authentication [PSA backend docs](#) will be a great reference to configure given provider.

CHAPTER 3

Sample usage

We provide a standalone test app for you to start easily, see how everything works with basic settings. It might be useful before integrating **djoser** into your backend application.

In this extremely short tutorial we are going to mimic the simplest flow: register user, log in and log out. We will also check resource access on each consecutive step. Let's go!

Clone repository and install **djoser** to your virtualenv:

```
$ git clone git@github.com:sunscrapers/djoser.git
$ cd djoser
$ pip install -e .
```

Go to the `testproject` directory, migrate the database and start the development server:

```
$ cd testproject
$ ./manage.py migrate
$ ./manage.py runserver 8088
```

Register a new user:

```
$ curl -X POST http://127.0.0.1:8088/auth/users/ --data 'username=djoser&
↪password=alpine12'
{"email": "", "username": "djoser", "id":1}
```

So far, so good. We have just created a new user using REST API.

Let's access user's details:

```
$ curl -LX GET http://127.0.0.1:8088/auth/users/me/
{"detail": "Authentication credentials were not provided."}
```

As we can see, we cannot access user profile without logging in. Pretty obvious.

Let's log in:

```
curl -X POST http://127.0.0.1:8088/auth/token/login/ --data 'username=djoser&
↳password=alpine12'
{"auth_token": "b704c9fc3655635646356ac2950269f352ea1139"}
```

We have just obtained an authorization token that we may use later in order to retrieve specific resources.

Let's access user's details again:

```
$ curl -LX GET http://127.0.0.1:8088/auth/users/me/
{"detail": "Authentication credentials were not provided."}
```

Access is still forbidden but let's offer the token we obtained:

```
$ curl -LX GET http://127.0.0.1:8088/auth/users/me/ -H 'Authorization: Token_
↳b704c9fc3655635646356ac2950269f352ea1139'
{"email": "", "username": "djoser", "id": 1}
```

Yay, it works!

Now let's log out:

```
curl -X POST http://127.0.0.1:8088/auth/token/logout/ -H 'Authorization: Token_
↳b704c9fc3655635646356ac2950269f352ea1139'
```

And try access user profile again:

```
$ curl -LX GET http://127.0.0.1:8088/auth/users/me/ -H 'Authorization: Token_
↳b704c9fc3655635646356ac2950269f352ea1139'
{"detail": "Invalid token"}
```

As we can see, user has been logged out successfully and the proper token has been removed.

Authentication Backends

Note: Both Token Based and JWT Authentication can coexist at same time. Simply, follow instructions for both authentication methods and it should work.

4.1 Token Based Authentication

Add 'rest_framework.authtoken' to INSTALLED_APPS:

```
INSTALLED_APPS = [  
    'django.contrib.auth',  
    (...),  
    'rest_framework',  
    'rest_framework.authtoken',  
    'djoser',  
    (...),  
]
```

Configure `urls.py`. Pay attention to `djoser.url.authtoken` module path:

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

Add `rest_framework.authentication.TokenAuthentication` to Django REST Framework authentication strategies tuple:

```
REST_FRAMEWORK = {  
    'DEFAULT_AUTHENTICATION_CLASSES': (  
        'rest_framework.authentication.TokenAuthentication',  
        ...  
    )  
}
```

(continues on next page)

(continued from previous page)

```
        (...)  
    ),  
}
```

Run migrations - this step will create tables for auth and authtoken apps:

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

4.2 JSON Web Token Authentication

4.2.1 Django Settings

Add `rest_framework_simplejwt.authentication.JWTAuthentication` to Django REST Framework authentication strategies tuple:

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

Configure *django-rest-framework-simplejwt* to use the *Authorization: JWT <access_token>* header:

```
SIMPLE_JWT = {  
    'AUTH_HEADER_TYPES': ('JWT',),  
}
```

4.2.2 urls.py

Configure `urls.py` with `djoser.url.jwt` module path:

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

You may optionally provide DJOSER settings:

```
DJOSER = {
    'PASSWORD_RESET_CONFIRM_URL': '#/password/reset/confirm/{uid}/{token}',
    'ACTIVATION_URL': '#/activate/{uid}/{token}',
    'SEND_ACTIVATION_EMAIL': True,
    'SERIALIZERS': {},
}
```

All following setting names written in CAPS are keys on DJOSER dict.

5.1 PASSWORD_RESET_CONFIRM_URL

URL to your frontend password reset page. It should contain {uid} and {token} placeholders, e.g. #/password-reset/{uid}/{token}. You should pass uid and token to reset password confirmation endpoint.

Required: True

5.2 SEND_ACTIVATION_EMAIL

If True user will be required to click activation link sent in email after:

- creating an account via `RegistrationView`
- updating his email via `UserView`

Default: False

5.3 SEND_CONFIRMATION_EMAIL

If `True`, register or activation endpoint will send confirmation email to user.

Default: `False`

5.4 ACTIVATION_URL

URL to your frontend activation page. It should contain `{uid}` and `{token}` placeholders, e.g. `#/activate/{uid}/{token}`. You should pass `uid` and `token` to activation endpoint.

Required: `True`

5.5 SET_USERNAME_RETYPE

If `True`, you need to pass `re_new_{{ User.USERNAME_FIELD }}` to `/{{ User.USERNAME_FIELD }}` endpoint, to validate username equality.

Default: `False`

5.6 SET_PASSWORD_RETYPE

If `True`, you need to pass `re_new_password` to `/password/` endpoint, to validate password equality.

Default: `False`

5.7 PASSWORD_RESET_CONFIRM_RETYPE

If `True`, you need to pass `re_new_password` to `/password/reset/confirm/` endpoint in order to validate password equality.

Default: `False`

5.8 LOGOUT_ON_PASSWORD_CHANGE

If `True`, setting new password will logout the user.

Default: `False`

5.9 USER_EMAIL_FIELD_NAME

Determines which field in `User` model is used for email in versions of Django before 1.11. In Django 1.11 and greater value of this setting is ignored and value provided by `User.get_email_field_name` is used. This setting will be dropped when Django 1.8 LTS goes EOL.

Default: `'email'`

5.10 PASSWORD_RESET_SHOW_EMAIL_NOT_FOUND

If `True`, posting a non-existent email to `/password/reset/` will return a `HTTP_400_BAD_REQUEST` response with an `EMAIL_NOT_FOUND` error message ('User with given email does not exist.').

If `False` (default), the `/password/reset/` endpoint will always return a `HTTP_204_NO_CONTENT` response.

Please note that setting this to `True` will expose information whether an email is registered in the system.

Default: `False`

5.11 TOKEN_MODEL

Points to which token model should be used for authentication. In case if only stateless tokens (e.g. JWT) are used in project it should be set to `None`.

Example: `'knox.models.AuthToken'` **Default:** `'rest_framework.authtoken.models.Token'`

5.12 SERIALIZERS

Dictionary which maps djoser serializer names to paths to serializer classes. This setting provides a way to easily override given serializer(s) - it's used to update the defaults, so by providing, e.g. one key, all the others will stay default.

Note: Current user endpoints now use the serializer specified by `SERIALIZERS['current_user']`. This enables better security and privacy: the serializers can be configured separately so that confidential fields that are returned to the current user are not shown in the regular user endpoints.

Examples

```
{
    'user': 'myapp.serializers.SpecialUserSerializer',
}
```

Default:

```
{
    'activation': 'djoser.serializers.ActivationSerializer',
    'password_reset': 'djoser.serializers.PasswordResetSerializer',
    'password_reset_confirm': 'djoser.serializers.PasswordResetConfirmSerializer',
    'password_reset_confirm_retype': 'djoser.serializers.
↪PasswordResetConfirmRetypeSerializer',
    'set_password': 'djoser.serializers.SetPasswordSerializer',
    'set_password_retype': 'djoser.serializers.SetPasswordRetypeSerializer',
    'set_username': 'djoser.serializers.SetUsernameSerializer',
    'set_username_retype': 'djoser.serializers.SetUsernameRetypeSerializer',
    'user_create': 'djoser.serializers.UserCreateSerializer',
    'user_delete': 'djoser.serializers.UserDeleteSerializer',
    'user': 'djoser.serializers.UserSerializer',
    'current_user': 'djoser.serializers.CurrentUserSerializer',
    'token': 'djoser.serializers.TokenSerializer',
    'token_create': 'djoser.serializers.TokenCreateSerializer',
}
```

5.13 EMAIL

Dictionary which maps djoser email names to paths to email classes. Same as in case of `SERIALIZERS` it allows partial override.

Examples

```
{
    'activation': 'myapp.email.AwesomeActivationEmail',
}
```

Default:

```
{
    'activation': 'djoser.email.ActivationEmail',
    'confirmation': 'djoser.email.ConfirmationEmail',
    'password_reset': 'djoser.email.PasswordResetEmail',
}
```

5.14 SOCIAL_AUTH_TOKEN_STRATEGY

String path to class responsible for token strategy used by social authentication.

Example: `'myapp.token.MyStrategy'` **Default:** `'djoser.social.token.jwt.TokenStrategy'`

5.15 SOCIAL_AUTH_ALLOWED_REDIRECT_URIS

List of allowed redirect URIs for social authentication.

Example: `['https://auth.example.com']` **Default:** `[]`

5.16 PERMISSIONS

Dictionary that maps permissions to certain views across Djoser.

Examples

```
{
    'user': ['djoser.permissions.CurrentUserOrAdminOrReadOnly']
}
```

Defaults

```
{
    'activation': ['rest_framework.permissions.AllowAny'],
    'password_reset': ['rest_framework.permissions.AllowAny'],
    'password_reset_confirm': ['rest_framework.permissions.AllowAny'],
    'set_password': ['djoser.permissions.CurrentUserOrAdmin'],
    'set_username': ['rest_framework.permissions.IsAuthenticated'],
    'user_create': ['rest_framework.permissions.AllowAny'],
    'user_delete': ['djoser.permissions.CurrentUserOrAdmin'],
}
```

(continues on next page)

(continued from previous page)

```
'user': ['djoser.permissions.CurrentUserOrAdminOrReadOnly'],
'user_list': ['djoser.permissions.CurrentUserOrAdminOrReadOnly'],
'token_create': ['rest_framework.permissions.AllowAny'],
'token_destroy': ['rest_framework.permissions.IsAuthenticated'],
}
```


CHAPTER 6

Base Endpoints

6.1 User

Use this endpoint to retrieve/update user.

Default URL: /users/me/ **Backward-compatible URL:** /me/

Method	Request	Response
GET	–	HTTP_200_OK <ul style="list-style-type: none">• {{ User. USERNAME_FIELD }}• {{ User._meta.pk. name }}• {{ User. REQUIRED_FIELDS }}
PUT	{{ User.REQUIRED_FIELDS }}	HTTP_200_OK <ul style="list-style-type: none">• {{ User. USERNAME_FIELD }}• {{ User._meta.pk. name }}• {{ User. REQUIRED_FIELDS }}

6.2 User Create

Use this endpoint to register new user. Your user model manager should implement `create_user` method and have `USERNAME_FIELD` and `REQUIRED_FIELDS` fields.

Default URL: `/users/` **Backward-compatible URL:** `/users/create/`

Method	Request	Response
POST	<ul style="list-style-type: none">• <code>{{ User. USERNAME_FIELD }}</code>• <code>{{ User. REQUIRED_FIELDS }}</code>• <code>password</code>	<code>HTTP_201_CREATED</code> <ul style="list-style-type: none">• <code>{{ User. USERNAME_FIELD }}</code>• <code>{{ User._meta.pk. name }}</code>• <code>{{ User. REQUIRED_FIELDS }}</code>

6.3 User Delete

Use this endpoint to delete authenticated user. By default it will simply verify password provided in `current_password`, delete the auth token if token based authentication is used and invoke delete for a given User instance. One of ways to customize the delete behavior is to override `User.delete`.

Default URL: `/users/me/`

Method	Request	Response
DELETE	<ul style="list-style-type: none">• <code>current_password</code>	<code>HTTP_204_NO_CONTENT</code> <code>HTTP_400_BAD_REQUEST</code> <ul style="list-style-type: none">• <code>current_password</code>

Backward-compatible URL: `/users/delete/`

Method	Request	Response
POST	<ul style="list-style-type: none">• <code>current_password</code>	<code>HTTP_204_NO_CONTENT</code> <code>HTTP_400_BAD_REQUEST</code> <ul style="list-style-type: none">• <code>current_password</code>

6.4 User Activate

Use this endpoint to activate user account. This endpoint is not a URL which will be directly exposed to your users - you should provide site in your frontend application (configured by `ACTIVATION_URL`) which will send POST request to activate endpoint.

Default URL: `/users/confirm/` **Backward-compatible URL:** `/users/activate/`

Method	Request	Response
POST	<ul style="list-style-type: none"> • <code>{{ User.USERNAME_FIELD }}</code> • <code>token</code> 	HTTP_204_NO_CONTENT

6.5 User Resend Activation E-mail

Use this endpoint to re-send the activation e-mail. Note that no e-mail would be sent if the user is already active or if they don't have a usable password. Also if the sending of activation e-mails is disabled in settings, this call will result in HTTP_400_BAD_REQUEST

Default URL: `/users/resend/`

Method	Request	Response
POST	<ul style="list-style-type: none"> • <code>{{ User.USERNAME_FIELD }}</code> 	HTTP_204_NO_CONTENT HTTP_400_BAD_REQUEST

6.6 Set Username

Use this endpoint to change user username (USERNAME_FIELD).

Default URL: `/users/change_username/` **Backward-compatible URL:** `/{{ User.USERNAME_FIELD }}/`

Note: `re_new_{{ User.USERNAME_FIELD }}` is only required if `SET_USERNAME_RETYPE` is True

Method	Request	Response
POST	<ul style="list-style-type: none"> • <code>new_{{ User.USERNAME_FIELD }}</code> • <code>re_new_{{ User.USERNAME_FIELD }}</code> • <code>current_password</code> 	HTTP_204_NO_CONTENT

6.7 Set Password

Use this endpoint to change user password.

Default URL: `/password/`

Note: `re_new_password` is only required if `SET_PASSWORD_RETYPE` is True

Method	Request	Response
POST	<ul style="list-style-type: none">• new_password• re_new_password• current_password	HTTP_204_NO_CONTENT

6.8 Reset Password

Use this endpoint to send email to user with password reset link. You have to setup `PASSWORD_RESET_CONFIRM_URL`.

Default URL: `/password/reset/`

Note: `HTTP_204_NO_CONTENT` if `PASSWORD_RESET_SHOW_EMAIL_NOT_FOUND` is `False`

Otherwise and if email does not exist in database `HTTP_400_BAD_REQUEST`

Method	Request	Response
POST	<pre>{{ User.USERNAME_FIELD }} }}</pre>	<ul style="list-style-type: none">• HTTP_204_NO_CONTENT• HTTP_400_BAD_REQUEST

6.9 Reset Password Confirmation

Use this endpoint to finish reset password process. This endpoint is not a URL which will be directly exposed to your users - you should provide site in your frontend application (configured by `PASSWORD_RESET_CONFIRM_URL`) which will send POST request to reset password confirmation endpoint.

Default URL: `/password/reset/confirm/`

Note: `re_new_password` is only required if `PASSWORD_RESET_CONFIRM_RETYPE` is `True`

Method	Request	Response
POST	<ul style="list-style-type: none">• <pre>{{ User.USERNAME_FIELD }}</pre>• token• new_password• re_new_password	HTTP_204_NO_CONTENT

Token Endpoints

7.1 Token Create

Use this endpoint to obtain user **authentication token**. This endpoint is available only if you are using token based authentication.

Default URL: /token/login/ **Backward-compatible URL:** /token/create/

Method	Request	Response
POST	<ul style="list-style-type: none">• {{ User. USERNAME_FIELD }}• password	HTTP_200_OK <ul style="list-style-type: none">• auth_token

7.2 Token Destroy

Use this endpoint to logout user (remove user authentication token). This endpoint is available only if you are using token based authentication.

Default URL: /token/logout/ **Backward-compatible URL:** /token/destroy/

Method	Request	Response
POST	—	HTTP_204_NO_CONTENT

8.1 JWT Create

Use this endpoint to obtain JWT.

Default URL: `/jwt/create/`

8.2 JWT Refresh

Use this endpoint to refresh JWT.

Default URL: `/jwt/refresh/`

8.3 JWT Verify

Use this endpoint to verify JWT.

Default URL: `/jwt/verify/`

Social Endpoints

Warning: This API is in beta quality - backward compatibility is not guaranteed in future versions and you may come across bugs.

9.1 Provider Auth

Using these endpoints you can authenticate with external tools.

The workflow should look like this:

1. Access the endpoint providing a `redirect_uri` that would perform the POST action later.
2. The request would return a JSON containing one key `authorization_url`. Redirect the user to that URL.
3. When the user authenticates with the external tool, that tool would redirect them to the `redirect_uri` you provided with a GET querystring containing two arguments: `code` and `state`
4. From the view that your user got redirected to, issue a POST request to the endpoint with the `code` and `state` arguments. You should use `application/x-www-form-urlencoded` not JSON. The user should be now authenticated in your application.

The list of providers is available at [social backend docs](#). please follow the instructions provided there to configure your backend.

Default URL: `/o/{{ provider }}/`

Note:

- `redirect_uri` is provided via GET parameters - not JSON
 - `state` parameter isn't always required e.g. in case of OpenID backends
-

Method	Request	Response
GET	<ul style="list-style-type: none">• <code>redirect_uri</code>	HTTP_200_OK <ul style="list-style-type: none">• <code>authorization_url</code> HTTP_400_BAD_REQUEST
POST	<ul style="list-style-type: none">• <code>code</code>• <code>state</code>	HTTP_201_CREATED <ul style="list-style-type: none">• <code>token</code> HTTP_400_BAD_REQUEST <ul style="list-style-type: none">• <code>non_field_errors</code>

Djoser provides a set of [signals](#) that allow you to hook into Djoser user management flow.

10.1 `user_registered`

This signal is sent after successful user registration.

Argument	Value
<code>sender</code>	sender class
<code>user</code>	user instance
<code>request</code>	request instance

At this point, `user` has already been created and saved.

10.2 `user_activated`

This signal is sent after successful user activation.

Argument	Value
<code>sender</code>	sender class
<code>user</code>	user instance
<code>request</code>	request instance

At this point, `user` has already been activated and saved.

11.1 Migrating from 1.3 to 1.4

Due to a lack of maintenance on the *django-rest-framework-jwt* project, Djoser has switched to using *django-rest-framework-simplejwt*. This update includes some backwards-incompatible changes:

1. The response from the JWT Create endpoint includes both an *access* and *refresh* token. *access* is essentially the same as the old *token* and can be used to authenticate requests. *refresh* is used to acquire a new access token.
2. The JWT Refresh endpoint requires the *refresh* token and returns a new *access* token.
3. The JWT Verify endpoint no longer returns *token*.
4. *django-rest-framework-simplejwt* uses *Authorization: Bearer <token>*. This can be overridden by adding the following to Django Settings:

```
SIMPLE_JWT = {  
    'AUTH_HEADER_TYPES': ('JWT',),  
}
```

11.2 Migrating from 1.1 to 1.2

There is no urgent need to change anything as backward compatibility is retained. That being said we ask you to change usage from old endpoints to new ones for the warm fuzzy feeling of being more RESTful :)

11.3 Migrating from 0.x to 1.0

The stable release has introduced a number of backward incompatible changes and purpose of this guide is to allow developer to quickly adapt a given project.

11.3.1 Removal of `UserEmailFactoryBase` and its subclasses

As mentioned in [Emails](#) page since 1.0 email support has been removed from Djoser and it is advised to use `django-templated-mail` for use cases which were previously handled by djoser email support. You can find out more about it in the [project documentation](#). Keep in mind that `DOMAIN` and `SITE_NAME` settings have also been moved to `django-templated-mail` as described in [settings](#) page.

11.3.2 Base URLs are no longer included with other URLs

Previously `djoser.urls.base` were bundled with `djoser.urls.authtoken`, however in some cases developer might not need them and therefore if base URLs are needed it is now necessary to explicitly include them, e.g.:

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

11.3.3 Dropped support for Django < 1.10

Support for Django 1.8 and 1.9 has been dropped in Django REST Framework 3.7 and hence there was no reason to keep it in djoser. It is recommended to upgrade to Django 1.11, since 1.10 will EOL in December 2017. [Django Deprecation Timeline](#) and [Django Release Notes](#) are very helpful in the process.

11.3.4 Some View class names and URLs has been updated

Also please note that for sake of consistency all URLs now end with a trailing slash. The trailing slash is optional to ensure compatibility with frontend tools that strip the trailing slash (eg Google's Chrome browser and Angular framework).

View class names:

- `RegistrationView` has been renamed to `UserCreateView`
- `LoginView` has been renamed to `TokenCreateView`
- `LogoutView` has been renamed to `TokenDestroyView`

Base URLs:

- `register/` has been renamed to `users/create/`
- `register` URL name has been renamed to `user-create`
- `activate/` has been renamed to `users/activate/`
- `activate` URL name has been renamed to `user-activate`

Token Based Authentication URLs:

- `login/` has been renamed to `token/create/`
- `login` URL name has been renamed to `token-create`
- `logout/` has been renamed to `token/destroy/`
- `logout` URL name has been renamed to `token-destroy`

CHAPTER 12

Emails

Explicit email support has been removed from djoser in 1.0.0. It didn't make sense to handle such responsibility in a package, which should simply provide an implementation of common authentication-related REST endpoints.

Email support is now handled with the [django-templated-mail](#) package.

Email classes can be overridden using [EMAIL setting](#)

Adjustment

If you need to customize any serializer behaviour you can use the `DJOSER['SERIALIZERS']` setting to use your own serializer classes in the built-in views. Or if you need to completely change the default djoser behaviour, you can always override djoser views with your own custom ones.

Define custom urls instead of reusing `djoser.urls`:

```
urlpatterns = patterns('',
    (...),
    url(r'^register/$', views.CustomRegistrationView.as_view()),
)
```

Define custom view/serializer (inherit from one of djoser class) and override necessary method/field:

```
class CustomRegistrationView(djoser.views.RegistrationView):

    def send_activation_email(self, *args, **kwargs):
        your_custom_email_sender(*args, **kwargs)
```

You could check djoser API in source code:

- `djoser.views`
- `djoser.serializers`

14.1 Early detecting invalid password reset tokens

When there is need to check if password reset token is still valid without actually resetting the password it is possible to approach the problem like so:

```
from django.contrib.auth.tokens import default_token_generator
from rest_framework import generics, permissions, status
from rest_framework.response import Response

from djoser import serializers

class PasswordTokenCheckView(generics.CreateAPIView):
    permission_classes = (
        permissions.AllowAny,
    )
    token_generator = default_token_generator
    serializer_class = serializers.UidAndTokenSerializer

    def post(self, request, *args, **kwargs):
        serializer = self.get_serializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        headers = self.get_success_headers(serializer.data)
        return Response(serializer.data, status=status.HTTP_200_OK, headers=headers)
```


CHAPTER 15

Indices and tables

- `genindex`
- `search`