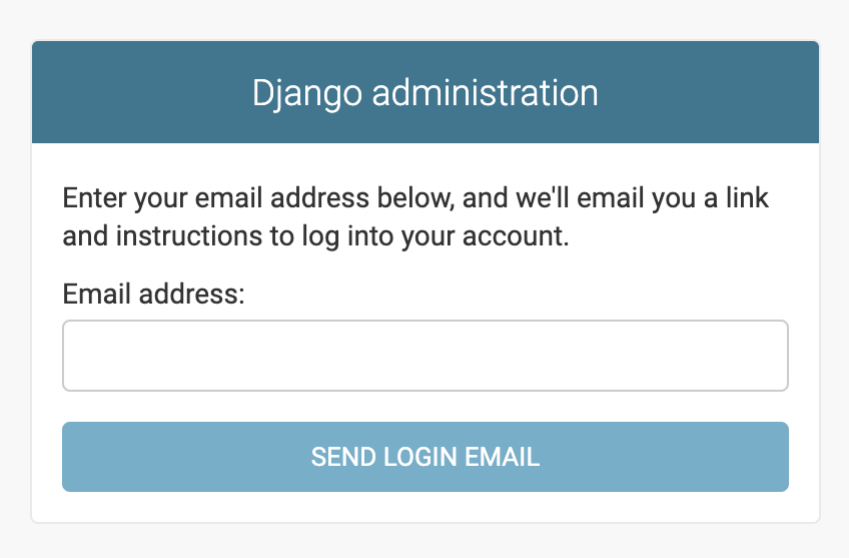
Django Mail Auth

Release 0.3.0

Sep 13, 2019

Contents:



Django Mail Auth is a lightweight authentication backend for Django, that does not require users to remember passwords.

Django Mail Auth features:

- custom user model support
- drop in Django admin support
- drop in Django User replacement
- extendable SMS support

This project was inspired by:

- Is it time for password-less login? by Ben Brown
- LOGIN WITHOUT PASSWORD MOST SECURE | WAIT.. WHAT? by Joris Snoek
- django-nopassword by Rolf Erik Lekang

CHAPTER 1

Installation

Run this command to install `django-mail-auth`:

```
pip install django-mail-auth
```

CHAPTER 2

Setup

First add *mailauth* to you installed apps:

```
INSTALLED_APPS = [
    # Django's builtin apps...

    'mailauth',
    'mailauth.contrib.admin',  # optional
    'mailauth.contrib.user',  # optional

    # other apps...
]
```

mailauth.contrib.admin is optional and will replace the admin's login with token based authentication too.

mailauth.contrib.user is optional and provides a new Django User model. The new User model needs to be enabled via the `AUTH_USER_MODEL` setting:

```
AUTH_USER_MODEL = 'mailauth_user.EmailUser'
```

Next you will need to add the new authentication backend:

```
AUTHENTICATION_BACKENDS = (
    # default, but now optional
    # This should be removed if you use mailauth.contrib.user or any other
    # custom user model that does not have a username/password
    'django.contrib.auth.backends.ModelBackend',

    # The new access token based authentication backend
    'mailauth.backends.MailAuthBackend',
)
```

Django's *ModelBackend* is only needed, if you still want to support password based authentication. If you don't, simply remove it from the list.

Last but not least, go to your URL root config *urls.py* and add the following:

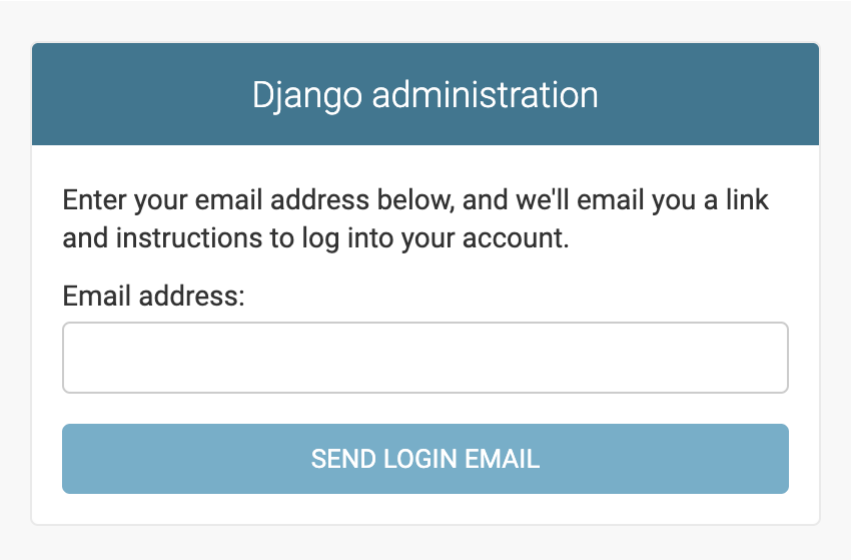
```
from django.urls import path


urlpatterns = [
    path('accounts/', include('mailauth.urls')),
]
```

That's it!

Note: Don't forget to setup you Email backend!

2.1 All Contents

2.1.1 Django Mail Auth



Django Mail Auth is a lightweight authentication backend for Django, that does not require users to remember passwords.

Django Mail Auth features:

- custom user model support
- drop in Django admin support
- drop in Django User replacement
- extendable SMS support

This project was inspired by:

- Is it time for password-less login? by Ben Brown
- LOGIN WITHOUT PASSWORD MOST SECURE | WAIT.. WHAT? by Joris Snoek

- django-nopassword by Rolf Erik Lekang

Installation

Run this command to install `django-mail-auth`:

```
pip install django-mail-auth
```

Setup

First add *mailauth* to you installed apps:

```
INSTALLED_APPS = [
    # Django's builtin apps...

    'mailauth',
    'mailauth.contrib.admin',  # optional
    'mailauth.contrib.user',  # optional

    # other apps...
]
```

mailauth.contrib.admin is optional and will replace the admin's login with token based authentication too.

mailauth.contrib.user is optional and provides a new Django User model. The new User model needs to be enabled via the `AUTH_USER_MODEL` setting:

```
AUTH_USER_MODEL = 'mailauth_user.EmailUser'
```

Next you will need to add the new authentication backend:

```
AUTHENTICATION_BACKENDS = (
    # default, but now optional
    # This should be removed if you use mailauth.contrib.user or any other
    # custom user model that does not have a username/password
    'django.contrib.auth.backends.ModelBackend',

    # The new access token based authentication backend
    'mailauth.backends.MailAuthBackend',
)
```

Django's *ModelBackend* is only needed, if you still want to support password based authentication. If you don't, simply remove it from the list.

Last but not least, go to your URL root config *urls.py* and add the following:

```
from django.urls import path


urlpatterns = [
    path('accounts/', include('mailauth.urls')),
]
```

That's it!

Note: Don't forget to setup you Email backend!

2.1.2 Templates

There are a couple relevant templates, that can be overridden to your needs.

Mail Auth templates

Login templates

`registration/login_requested.html`

This template will be displayed after a user successfully requested a login URL. This template is not proved by the package and needs to be created.

Email templates

`registration/login_subject.txt`

This template defines the subject line of the email that will be sent to the user.

This template is provided by the package and can be overridden.

`registration/login_email.txt`

This is the plain text template for the email containing the authentication URL that will be sent to the user.

This template is provided by the package and can be overridden.

`registration/login_email.html`

This is the HTML template for the email containing the authentication URL that will be sent to the user.

This template is optional. If not provided, only plain text emails will be sent.

Django related templates

Mail Auth uses Django's default templates for the login views.

Login templates

`registration/login.html`

This template displays login form, where a user can request a login URL. This template is not proved Django or by the package and needs to be created.

`registration/logged_out.html`

This template will be displayed after a successful logout. This template is not proved Django or by the package and needs to be created.

2.1.3 Customizing

Django Mail Auth can be easily extend. Besides template adaptations it is possible to send send different messages like SMS. To make those changes you will need to write a custom login form.

Custom login form

Custom login forms need to inherit from *BaseLoginForm* and override the *save* method.

The following example is for a login SMS via twilio. This will require a custom user model with a unique `phone_number` field:

```
from django import forms
from django.contrib.auth import get_user_model
from django.template import loader
from mailauth.forms import BaseLoginForm


class SmsLoginForm(BaseLoginForm):
    phone_number = forms.CharField()

    template_name = 'registration/login_sms.txt
    from_number = None

    def __init__(self, *args, **kwargs):
        self.twilio_client = TwilioRestClient(
            settings.TWILIO_SID,
            settings.TWILIO_AUTH_TOKEN
        )
        super().__init__(*args, **kwargs)

    def save(self):
        phone_number = self.cleaned_data['phone_number']
        user = get_user_model().objects.get(
            phone_number=phone_number
        )
        context = self.get_context(self.request, user)

        from_number = self.from_number or getattr(
            settings, 'DEFAULT_FROM_NUMBER'
        )
        sms_content = loader.render_to_string(
            self.template_name, context
        )

        self.twilio_client.messages.create(
            to=user.phone_number,
            from_=from_number,
            body=sms_content
        )
```

To add the new login form, simply add a new login view to your URL config with the custom form:

```
from django.urls import path
from mailauth.views import LoginView

from .forms import SmsLoginForm
```

(continues on next page)

(continued from previous page)

```
urlpatterns = [
    path(
        'login/sms/',
        LoginView.as_view(form_class=SmsLoginForm),
        name='login-sms'
    ),
]
```

API documentation

class mailauth.forms.**BaseLoginForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

get_context(*request*, *user*)
Return the context for a message template render.

Parameters

- **request** (*django.http.request.HttpRequest*) – Current request.
- **user** – The user requesting a login message.

Returns

A context dictionary including:

- site
- site_name
- token
- login_url
- user

Return type dict

get_login_url(*request*, *token*, *next=None*)
Return user login URL including the access token.

Parameters

- **request** (*django.http.request.HttpRequest*) – Current request.
- **token** (*str*) – The user specific authentication token.
- **next** (*str*) – The path the user should be forwarded to after login.

Returns User login URL including the access token.

Return type str

get_token(*user*)
Return the access token.

save()
Send login URL to users.

Called from the login view, if the form is valid.

This method must be implemented by subclasses. This method should trigger the login url to be sent to the user.

2.1.4 Settings

Mail Auth settings

LOGIN_URL_TIMEOUT
Default: `900`

Defines how long a login code is valid in seconds.

LOGIN_REQUESTED_URL
Default: `accounts/login/success`

Defines the URL the user will be redirected to, after requesting an authentication message.

LOGIN_TOKEN_SINGLE_USE
Default: `True`

Defines if a token can be used more than once. If `True`, the same token can only be used once and will be invalid the next try. If `False`, the same token can be used multiple times and remains valid until expired.

Django related settings

DEFAULT_FROM_EMAIL
Default: `'root@example.com'`

The sender email address for authentication emails send by Django Mail Auth.

SECRET_KEY

> **Attention:** *Keep it secret, keep it safe!*
>
> This key is the foundation of all of Django security measures and for this package.

2.1.5 Contributing

To install the development requirements simply run:

```
python setup.py develop
```

To run test suite run:

```
python setup.py test
```

. . . and to run the entire test suite, simply use tox:

```
pip install --upgrade tox
tox
```

To build the documentation run:

```
python setup.py build_sphinx
open docs/_build/html/index.html
```

The sample app

To run a full example — e.g. to debug frontend code – you can run:

```
python setup.py develop
python tests/testapp/manage.py migrate
python tests/testapp/manage.py createsuperuser
# You will be asked for the email address of your new superuser
python tests/testapp/manage.py runserver
```

Next you can go to https://localhost:8000/admin/ and log in with your newly created superuser.

Index