Django Mail Auth

Release 3.1.4

Oct 17, 2022
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
- drop in Wagtail login 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
Run this command to install `django-mail-auth`:

```
python3 -m pip install django-mail-auth[wagtail]
```
First add `mailauth` to your installed apps:

```python
INSTALLED_APPS = [
    # Django's built-in apps...
    'mailauth',
    'mailauth.contrib.admin',  # optional
    'mailauth.contrib.user',   # optional
    # optional, must be included before "wagtail.admin"
    'mailauth.contrib.wagtail',
    # 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:

```python
# This setting should be either "EmailUser" or any custom subclass of "AbstractEmailUser"
AUTH_USER_MODEL = 'mailauth_user.EmailUser'

# optional, Wagtail only
WAGTAILUSERS_PASSWORD_ENABLED = False
```

Next you will need to add the new authentication backend:

```python
AUTHENTICATION_BACKENDS = (
    # default, but now optional
    # This should be removed if you use mailauth.contrib.user or any other
```
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:

```python
from django.urls import path
urlpatterns = [
    path('accounts/', include('mailauth.urls')),
    # optional, must be before "wagtail.admin.urls"
    path('', include('mailauth.contrib.wagtail.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
- drop in Wagtail login 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:

```bash
python3 -m pip install django-mail-auth[wagtail]
```

Setup

First add mailauth to you installed apps:

```python
INSTALLED_APPS = [
    # Django's builtin apps...
    'mailauth',
    'mailauth.contrib.admin', # optional
    'mailauth.contrib.user', # optional
    # optional, must be included before "wagtail.admin"
    'mailauth.contrib.wagtail',

    # 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:

```python
# This setting should be either "EmailUser" or
# any custom subclass of "AbstractEmailUser"
AUTH_USER_MODEL = 'mailauth_user.EmailUser'

# optional, Wagtail only
WAGTAILUSERS_PASSWORD_ENABLED = False
```
Next you will need to add the new authentication backend:

```python
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:

```python
from django.urls import path

urlpatterns = [
    path('accounts/', include('mailauth.urls')),
    # optional, must be before "wagtail.admin.urls"
    path('', include('mailauth.contrib.wagtail.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 Privacy

Anonymization

User privacy is important, not only to meet local regulations, but also to protect your users and allow them to exercise
their rights. However, it’s not always practical to delete users, especially if they have dependent objects, that are
relevant for statistical analysis.

Anonymization is a process of removing the user’s personal data whilst keeping related data intact. This is done by
using the anonymize method.

AbstractEmailUser.anonymize(commit=True)

Anonymize the user data for privacy purposes.

This method will erase the email address, first and last name. You may overwrite this method to add additional
fields to anonymize:

```python
class MyUser(AbstractEmailUser):
    def anonymize(self, commit=True):
        super().anonymize(commit=False) # do not commit yet
        self.phone_number = None
        if commit:
            self.save()
```

This method may be overwritten to provide anonymization for you custom user model.

Related objects may also listen to the anonymize signal.

mailauth.contrib.user.signals.anonymize

All those methods can be conveniently triggered via the anonymize admin action.

2.1. All Contents
class mailauth.contrib.user.admin.AnonymizableAdminMixin
    Bases: object

    Mixin for admin classes that provides a anonymize action.
    This mixin calls the anonymize method of all user model instances.

Liability Waiver

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2.1.4 Customizing

Custom login message (like SMS)

Django Mail Auth can be easily extended. Besides template adaptations it is possible to 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:

```python
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
        )
```

(continues on next page)
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:

```python
from django.urls import path
from mailauth.views import LoginView
from .forms import SmsLoginForm

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

**API documentation**

class mailauth.forms.BaseLoginForm

Bases: django.forms.forms.Form

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_mail_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_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.

**Custom User Model**

For convenience, Django Mail Auth provides a `EmailUser` which is almost identical to Django’s built in `User` but without the `password` and `username` field. The `email` field serves as a username and is – different to Django’s `User` – unique and case insensitive.

**Implementing a custom User model**

```python
from mailauth.contrib.user.models import AbstractEmailUser
from phonenumber_field.modelfields import PhoneNumberField

class SMSUser(AbstractEmailUser):
    phone_number = phone = PhoneNumberField(_('phone number'), unique=True, db_index=True)

class Meta(AbstractEmailUser.Meta):
    verbose_name = _('user')
    verbose_name_plural = _('users')
    swappable = "AUTH_USER_MODEL"
```

**Note:** Do not forget to adjust your `AUTH_USER_MODEL` to correct `app_label.ModelName`. 

API documentation

class mailauth.contrib.user.models.AbstractEmailUser(*args, **kwargs)
   Bases: django.contrib.auth.models.AbstractUser

   email
      Unique and case insensitive to serve as a better username.

   session_salt
      Salt for the session hash replacing the password in this function.

   anonymize(commit=True)
      Anonymize the user data for privacy purposes.

      This method will erase the email address, first and last name. You may overwrite this method to add additional fields to anonymize:

      class MyUser(AbstractEmailUser):
         def anonymize(self, commit=True):
            super().anonymize(commit=False) # do not commit yet
            self.phone_number = None
            if commit:
               self.save()

get_session_auth_hash()
   Return an HMAC of the session_salt field.

has_usable_password()
   Return False if set_unusable_password() has been called for this user.

session_salt
   Salt for the session hash replacing the password in this function.

class mailauth.contrib.user.models.EmailUser(id, last_login, is_superuser, first_name,
                                           last_name, is_staff, is_active, dateJoined, email, session_salt)
   Bases: mailauth.contrib.user.models.AbstractEmailUser

exception DoesNotExist
   Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
   Bases: django.core.exceptions.MultipleObjectsReturned

2.1.5 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.

### 2.1.6 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/](https://localhost:8000/admin/) and log in with your newly created superuser.
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
</table>
| **A**
| AbstractEmailUser (class in mailauth.contrib.user.models), 13 | LOGIN_TOKEN_SINGLE_USE, 13 |
| AnonymizableAdminMixin (class in mailauth.contrib.user.admin), 9 | LOGIN_URL_TIMEOUT, 13 |
| anonymize (in module mailauth.contrib.user.signals), 9 | S |
| anonymize() (mailauth.contrib.user.models.AbstractEmailUser method), 13 | save() (mailauth.forms.BaseLoginForm method), 12 |
| **B**
| BaseLoginForm (class in mailauth.forms), 11 | SECRET_KEY, 14 |
| **D**
| DEFAULT_FROM_EMAIL, 14 |
| **E**
| EmailUser (class in mailauth.contrib.user.models), 13 |
| EmailUser.DoesNotExist, 13 |
| EmailUser.MultipleObjectsReturned, 13 |
| **G**
| get_login_url() (mailauth.forms.BaseLoginForm method), 11 |
| get_mail_context() (mailauth.forms.BaseLoginForm method), 11 |
| get_session_auth_hash() (mailauth.contrib.user.models.AbstractEmailUser method), 13 |
| get_token() (mailauth.forms.BaseLoginForm method), 12 |
| **H**
| has_usable_password() (mailauth.contrib.user.models.AbstractEmailUser method), 13 |
| **L**
| LOGIN_REQUESTED_URL, 13 |