django-newauth Documentation

Release 0.37

BeProud

May 19, 2017

Contents

django-newauth (newauth for short) is a Django application developed at BeProud Inc. that implements authentication the right way. With newauth you can make user models that look like this:

```
from django.db import models

from newauth.api import UserBase

class User(UserBase):
    full_name = models.CharField(u"Full Name", max_length=255)
    email = models.EmailField('Email Address')
    profile = models.TextField('Profile Bio', blank=True, null=True)
    avatar = models.ImageField('Avatar', upload_to='profileimg/', blank=True,␣
↪null=True)

    def get_display_name(self):
        return self.full_name

    class Meta:
        db_table = 'my_user_table'
        verbose_name = u"Djangonaut"
        verbose_name_plural = u"Djangonaut"
```

Table of contents:

CHAPTER 1

Overview

newauth is a kind of authentication framework that allows you to customize user models and use the customized user models throughout your application. A number of tools are provided to allow you to authenticate those customized user objects easily. newauth has the following features:

Customizable User Models

The Django auth contrib application has a User model which contains a number of fields. New fields cannot be added so you must create a seperate model with a one to one relationship to the user and retrieve that model object using the get_profile() method or via the related field of a OneToOneField (i.e. user.myprofile).

However, since the User model cannot be modified, you are stuck with all of the fields present there even if you don't use them. You are also stuck with using the integer key that comes with a User model. You are out of luck if you would like to have a User with a UID for a key or have more users than fit into an integer.

Instead of a given User model, newauth provides a *`UserBase`* abstract model class which you can extend in your application. *`UserBase`* objects don't have any fields and newauth only depends on the fact that a `pk` exists. The id is left to be defined by the subclass in your own user defined application.

Multiple Auth Backends

newauth supports authentication backends much like Django's contrib.auth application. However, newauth attaches a name to backends and allows specifying which backend to use to log the user in. This allows fine grained authentication checking for a specific type of user.

Auth API and Tools

newauth provides an API much like Django's contrib.auth module which allows you to login and logout users programatically. It also provides decorators like the *`login_required()`* decorator.

Storage Backends

newauth allows you to specify how to store information about the logged in user. When a user logs in it is necessary to store the user's pk and backend in order to get the user's information on subsequent requests.

Django's contrib.auth application relies on Django sessions to store this information. However, newauth allows for the use of storage backends other than sessions such as secure cookies, much like the Django messages framework.

CHAPTER 2

Installation

Application Install

Setting up django-newauth is easy.

First install newauth using pip (doesn't work yet):

```
$ pip install django-newauth
```

Next, add `newauth` to INSTALLED_APPS in your `settings.py`.

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    #...
    'newauth',
    #...
)
```

Middleware Setup

You need to install the authentication middleware to support authentication. Add `newauth.middleware.AuthMiddleware` to the MIDDLEWARE_CLASSES setting in your `settings.py`.

```
MIDDLEWARE_CLASSES = (
    #...
    'newauth.middleware.AuthMiddleware',
    #...
)
```

From here you should proceed to the *Tutorial*.

CHAPTER 3

Tutorial

newauth is a framework for building your own user account modules within your Django site. Following this tutorial you can get a feel for how to use bpauth but if you like you can jump to the other parts of the tutorial here.

Getting Started

In this tutorial we will get aquainted with how to use bpauth. bpauth provides many utilities to use to manage authentication and user data. Reading the source can be a bit daunting so we'll go through the most basic usage here.

First we will start by creating a Django application named "account".

```
$ python manage.py startapp account
```

Within the account application we will create the user model for our site. On this model we can add whatever fields might be useful to our application. The User class will extend from bpauth's *UserBase* base model class.

Lets edit the models.py:

```
from django.db import models
from newauth.api import UserBase

class MyUser(UserBase):
    email = models.EmailField('Email Address')
    profile = models.TextField('Profile Bio', blank=True, null=True)
    avatar = models.ImageField('Avatar', upload_to='profileimg/', blank=True,␣
↪null=True)
```

Here we have a basic user class for our application. All of the fields we want can be attached directly on the user object. Next we'll want to make an anonymous user class that has the same fields as our user class:

```
from django.db import models
from newauth.api import AnonymousUserBase

class MyAnonymousUser(AnonymousUserBase):
```

```
    email = None
    profile = None
    avatar = None
```

Now let's write a simple view.

```
from django.views.generic.simple import direct_to_template

from account.models import User

def my_profile(request):
    return direct_to_template(request, 'account/my_profile.html', {
        'avatar': request.auth_user.avatar,
        'profile': request.auth_user.profile,
    })
```

You see here we can access all of the fields pertaining to the user's profile directly on the user object. This is very convenient as you do not have to use 'get_profile()' to get access to added fields for your user as you have to do with Django's contrib.auth application.

The UserBase model defines a number of methods on it to allow third party applications to interact with your model. Because third party applications cannot assume you have any fields on your model besides the 'pk' field, third party applications need to interact with your MyUser model via these overridable methods.

Here we override the *get_display_name()* method to use the first part of the user's email address as their display name. By default the *UserBase* class simply returns the user's primary key.

```
class MyUser(UserBase):

    # ...

    def get_display_name(self):
        return self.email.split('@')[0]
```

The *UserBase* class also defines a *get_real_name()* method that can be used for internal, editing, and administrative functions where a real name would be more appropriate, such as invoicing etc.

In *the next section* we'll discuss how to use newauth to create user models that use simple username/password authentication.

Basic Username/Password Authentication

newauth provides a `basic` submodule that provides some basic functionality for creating account applications with user models that use basic username/password authentication. It is a good example of how to use newauth to create your own user module.

The *BasicUserBase* model extends the *UserBase* model and adds a username and password field and implements password checking. Here we can create a MyUser model as we did before but with username/password functonality.

```
from django.db import models
from newauth.api import BasicUserBase

class MyUser(BasicUserBase):
    email = models.EmailField('Email Address')
    profile = models.TextField('Profile Bio', blank=True, null=True)
    avatar = models.ImageField('Avatar', upload_to='profileimg/', blank=True,␣
↪null=True)
```

Now we can use the included *BasicUserBackend* and BasicAuthForm <newauth.forms.BasicAuthForm in combination with the *login()* view to authenticate our user.

Here we'll set up the *BasicUserBackend* in settings.py:

```
NEWAUTH_BACKENDS = {
    'default': {
        'backend': 'newauth.backends.BasicUserBackend',
        'user': 'account.models.MyUser',
        'anon_user': 'account.models.MyAnonymousUser',
    }
}
```

The *login()* view uses the *BasicAuthForm* by default but we can tell it to use the *BasicAuthForm* explicitly.

```
from django.conf.urls import url
from django.conf import settings

from newauth.views import login
from newauth.forms import BasicAuthForm

urlpatterns = [
    url(r'^login$', login, name='newauth_login', kwargs={
        'authentication_form': BasicAuthForm,
    }),
]
```

We can also use the provided BasicUserAdmin to add functionality to Django's admin. The BasicUserAdmin class implements creating new users and password change in much the same way as Django's auth application. This makes it very easy to implement usable admin pages:

```
from django.contrib import admin
from beproud.django.auth.basic.admin import BasicUserAdmin

from account.models import MyUser

admin.site.register(MyUser, BasicUserAdmin)
```

In *the next section* we'll discuss how to set up the login and logout urls.

Setting up the Login and Logout Views

In this section we'll set up the provided login and logout views to allow for logging in using basic username and password authentication.

URL Routing

You can add the login and logout views to your url configuration by including the newauth.urls module.

```
urlpatterns = [
    # ...
    url(r'^account/', include('newauth.urls')),
    # ...
]
```

Customizing the login form

The provided login view can use any form class that extends the *BaseAuthForm* class. You can specify the form class by passing it in the `authentication_form` argument to the `login <newauth.views.login()` view.

```
urlpatterns = [
    # ...
    url(r'^login/$', 'newauth.views.login', name='newauth_login', kwargs={
        'authentication_form': MyLoginForm,
    }),
    # ...
]
```

In *the next section* we'll discuss how to limit access to views to logged-in users.

Limiting access to logged-in users

The raw way

The simple, raw way to limit access to pages is to check *request.user.is_authenticated()* and either redirect to a login page or display an error message. This is largely based off of django's User.is_authenticated() method.

```
from django.http import HttpResponseRedirect

def my_view(request):
    if not request.user.is_authenticated():
        return HttpResponseRedirect('/login/?next=%s' % request.path)
    # ...
```

The login_required decorator

You can limit access to logged-in users using the *login_required()* decorator. The login_required decorator can be used in the same way that the Django login_required decorator can be used but with some notable differences.

login_required() can take no arguments in the same way that the login_required() decorator for django auth does.

```
from newauth.decorators import login_required

@login_required
def my_view(request):
    ...
```

It also takes the same keyword arguments.

```
from newauth.decorators import login_required

@login_required(login_url="/mylogin", redirect_field_name="next_url")
def my_view(request):
    ...
```

However it also can take a list of backend names so that you can specify the specific backends that are required to execute that view.

```
from newauth.decorators import login_required

@login_required(["default", "backend2"])
def my_view(request):
    ...
```

We'll tie everything we have set up so far in *the next section* by adding the settings to make a working example.

Tieing it all together

Now we'll tie it all together by setting the appropriate settings in our settings.py.

As we mentioned in the *Installation* you'll need to add `newauth` to your INSTALLED_APPS:

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    #...
    'newauth',
    #...
)
```

and add the *AuthMiddleware* to your MIDDLEWARE_CLASSES

```
MIDDLEWARE_CLASSES = (
    #...
    'newauth.middleware.AuthMiddleware',
    #...
)
```

Here we will set up the *NEWAUTH_BACKENDS* so that we can authenticate and use our MyUser class that we set up earlier.

```
NEWAUTH_BACKENDS = {
    'default': {
        'backend': (
            'newauth.backend.BasicUserBackend',
        ),
        'user': 'account.models.MyUser',
        'anon_user': 'account.models.MyAnonymousUser',
    }
}
```

And that's it! You should have a simple working example of how to use newauth. newauth allows a lot of customization so you can continue on and read some of the more advanced documentation or play some more with the example.

Good Luck!

CHAPTER 4

Writing Auth Backends

newauth provides a couple base classes for defining backends. All backends should extend the BaseAuthBackend. Backend classes are created in much the same way as you would for the django.contrib.auth app. You need to create an `authenticate()` method that takes the arguments (credentials) that your backend requires and a `get_user()` method that takes an id and returns an instance of the user.

Here is a very simple backend that authenticates a user using only their user id. Not a very secure authentication but it illustrates how to write an authentication backend:

```
from nweauth.backend import BaseAuthBackend
from newauth.models import User

class UserIdBackend(BaseAuthBackend):
    def authenticate(self, user_id):
        return self.get_user(user_id)

    def get_user(self, user_id):
        try:
            return User.objects.get(pk=user_id)
        except User.DoesNotExist, e:
            return None
```

Using Model Authentication

Most of the time, however, you will be creating models extending the *`UserBase`* class so you can extend the provided `ModelAuthBackend` which provides a default implementation of `get_user()` which retrieves an instance of the model associated with the backend in the *`NEWAUTH_BACKENDS`* in your settings.py.

Extending Username/Password Authentication

CHAPTER 5

Authentication Forms

newauth provides a *BaseAuthForm* which can be used to authenticate users. You can simply implement the *get_credentials()* method on the form and add the needed fields. You can also override the `auth_failure` key in the `default_error_messages` dictionary property to provide your own error message.

```
from django import forms
from django.utils.translation import ugettext_lazy as _
from newauth.forms import BaseAuthForm

class AuthForm(BaseAuthForm):
    email = forms.EmailField()
    password = forms.CharField(widget=forms.PasswordInput)

    default_error_messages = {
        'auth_failure': _("Please enter the correct email and password."),
    }

    def get_credentials(self):
        return {
            'email': self.cleaned_data['email'],
            'password': self.cleaned_data['password'],
        }
```

The authenticated user can be obtained by calling the *get_user()* method in views after calling the `is_valid()` method. Here is an example of a **very** simple example of a login view:

```
from django.shortcuts import redirect
from newauth.api import login

from account.forms import AuthForm

def mylogin(request):
    form = AuthForm(request.POST or None)
    if request.method == 'POST':
        if form.is_valid():
            user = form.get_user()
```

```
            # Login the user
            login(request, user)
            return redirect('/')
    else:
        return ("""<html><body>"""
                """<form action="" method="POST">%s</form>"""
                """</body></html>""") % form
```

In the next section we'll discuss how to limit access to views to logged-in users.

CHAPTER 6

Using User objects with third party apps

newauth has the concept of a "default" user model. This model can be imported via the newauth.models module.

```
from django.db import models
from newauth.models import User

class ThirdPartyModel(models.Model):
    user = models.ForeignKey(User)
    #...
```

CHAPTER 7

API Reference

Release 0.37

Date May 19, 2017

newauth.admin

class newauth.admin.**BasicUserCreationForm**(**args*, ***kwargs*)
A form that creates a user, with no privileges, from the given username and password.

class newauth.admin.**PasswordChangeForm**(*user*, **args*, ***kwargs*)
A form used to change the password of a user in the admin interface.

save(*commit=True*)
Saves the new password.

newauth.api

Alex Gaynor will kill me

class newauth.api.**UserBase**(**args*, ***kwargs*)
Base User class

Primary key can be defined in sub classes. This class makes no assumptions about the format of the primary key. Only a pk property (the primary key might be something other than id) and the methods implemented below can be assumed are present.

This class also makes no assumptions about underlying implementations like what fields are on the User object or the table name.

get_display_name()
Name for display. Usually a username or something similar. Usually used for public pages. This method should return a unicode object.

get_real_name()
The user's real name or closest approximation. Usually used for private pages etc.

is_authenticated()
Returns whether a user is authenticated or not. The default is for this method to always return true for subclasses of UserBase and False for subclasses of AnonymousUserBase

class newauth.api.**AnonymousUserBase**
A simple anonymous user.

class newauth.api.**BasicUserManager**
A default manager that can be used with models that subclass BasicUserBase.

class newauth.api.**BasicUserBase**(**args*, ***kwargs*)
A basic user that is authenticated with a username and password.

This class can be subclassed when using a simple username password auth system.

check_password(*raw_password*)
Returns a boolean of whether the raw_password was correct. Handles encryption formats behind the scenes.

set_password(*raw_password*)
Sets the password of the user. Alorithm

newauth.api.**authenticate**(*_backend_name=None*, ***credentials*)
Authenticate a user with using the available auth backends. If a _backend_name is provided require authentication via the backend with that name.

If the given credentials are valid, return a User object as provided by the backend.

newauth.api.**login**(*request*, *user*, *backend_name=None*)
Log the user in given the request object. This will save any data needed to auth the user (user id and backend name) using the auth session storage (not necessarily django sessions).

newauth.api.**logout**(*request*)
Logs the user out. This will clear the user's login session data from the auth session storage.

newauth.backend

class newauth.backend.**BasicUserBackend**(*backend_name*)
A simple backend to authenticate any subclass of BasicUserBase.

newauth.decorators

newauth.decorators.**login_required**(*backend_list=None*, *login_url=None*, *redirect_field_name='next'*)
Decorator for views that checks that the user is logged in, redirecting to the log-in page if necessary.

newauth.forms

class newauth.forms.**BaseAuthForm**(**args*, ***kwargs*)
A base authentication form. Authentication forms can subclass this form and implement the get_credentials() method which will return a dictionary of credentials from self.cleaned_data that can be used with authenticate()

get_credentials()
Gets credentials as a dict object from self.cleaned_data

get_user()
Gets the cached user object for the user that logged in using this form.

class newauth.forms.**BasicAuthForm**(**args*, ***kwargs*)
A basic authentication form that will authenticate a user using a username and password. Useful for subclasses of BasicUser.

newauth.middleware

class newauth.middleware.**AuthMiddleware**(*get_response=None*)
Middleware for getting the current user object and attaching it to the request.

newauth.models

newauth.models.**get_user_model**(*model_name=None*)
Used to get access to defined user models.

from newauth.models import User, get_user_model

OtherUser = get_user_model('other')

class MyProfile(models.Model): user = models.ForeignKey(User) other_user_type = models.ForeignKey(OtherUser)

newauth.views

newauth.views.**login**(*request*, **args*, ***kwargs*)
Displays the login form and handles the login action.

newauth.views.**logout**(*request*, *next_page=None*, *template_name='registration/logged_out.html'*, *redirect_field_name='next'*)
Logs out the user and displays 'You are logged out' message.

CHAPTER 8

`settings` – newauth Settings

`settings`.**NEWAUTH_BACKENDS**
TODO

`settings`.**NEWAUTH_USER_PROPERTY**
TODO

`settings`.**NEWAUTH_USER_SESSION_STORAGE**
TODO

`settings`.**NEWAUTH_PASSWORD_ALGO**
TODO

CHAPTER 9

ChangeLog

Release 0.37 (2017-05-15)

- Support Python-3.6
- Support Django-1.11
- Drop Django-1.9

Release 0.36 (2016-12-14)

- Fixed: issue #7 Exception Type: AttributeError at /logout/

Release 0.35 (2016-11-30)

- 'newauth.middleware.AuthMiddleware' supports `settings.MIDDLEWARE` since django-1.10

Release 0.34 (2016-11-25)

- Support Django-1.10
- Drop Django-1.7 or earlier
- Provide documentation on ReadTheDocs: http://django-newauth.rtfd.io/

Release 0.33 (2016-02-01)

- More support Django-1.8 & 1.9

Release 0.32 (2016-01-19)

- Support Django-1.8 & 1.9

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

s

Index