Django Log In with Email not Username

By Will Vincent   Jul 13, 2020   0 Comments

Django was first released in 2005 and since then a lot has changed in web development, notably the predominant pattern at the time of using username/email/password has been simplified to just email/password. However, due to legacy reasons around the built-in Django User model, it can take a few extra steps to implement this change in practice.

In this tutorial we'll walk through how to implement email and password only for sign up and log in on a new Django project using a custom user model and the popular django-allauth package.

Getting Started

I'll assume you know how to start and install a new Django project using Pipenv. If not, please see here for additional help.

On the command line, create a new directory for our code called `code` (it can live anywhere on your computer but we'll use the Desktop as I'm on a Mac). Then install Django, start the virtual environment with `shell`, and create a new project.

```
$ cd ~/Desktop
$ mkdir code && cd code
$ pipenv install django~=3.0.0
$ pipenv shell
(code) $ django-admin startproject config .
```

Ok, we *could* just jump into it here but I'm done, done, done with writing any more Django tutorials that don't use a custom user model upfront. So, let's configure a custom user model before running `migrate` for the first time.

Custom User Model

I wrote about this in length over here so I'm just going to give the commands in this post.

Create a `users` app.

```
(code) $ python manage.py startapp users
```

Update the `settings.py` file.

```
# config/settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # Local
    'users.apps.UsersConfig', # new
]
...
AUTH_USER_MODEL = 'users.CustomUser' # new
```

Create our `CustomUser` model.

```
# users/models.py
from django.contrib.auth.models import AbstractUser
from django.db import models

class CustomUser(AbstractUser):
    # add additional fields in here

    def __str__(self):
        return self.email
```

Create a new `users/forms.py` file and then fill it with the following code.

```
# users/forms.py
from django import forms
from django.contrib.auth.forms import UserCreationForm, UserChangeForm
from .models import CustomUser

class CustomUserCreationForm(UserCreationForm):

    class Meta:
        model = CustomUser
        fields = ('username', 'email')

class CustomUserChangeForm(UserChangeForm):

    class Meta:
        model = CustomUser
        fields = ('username', 'email')
```

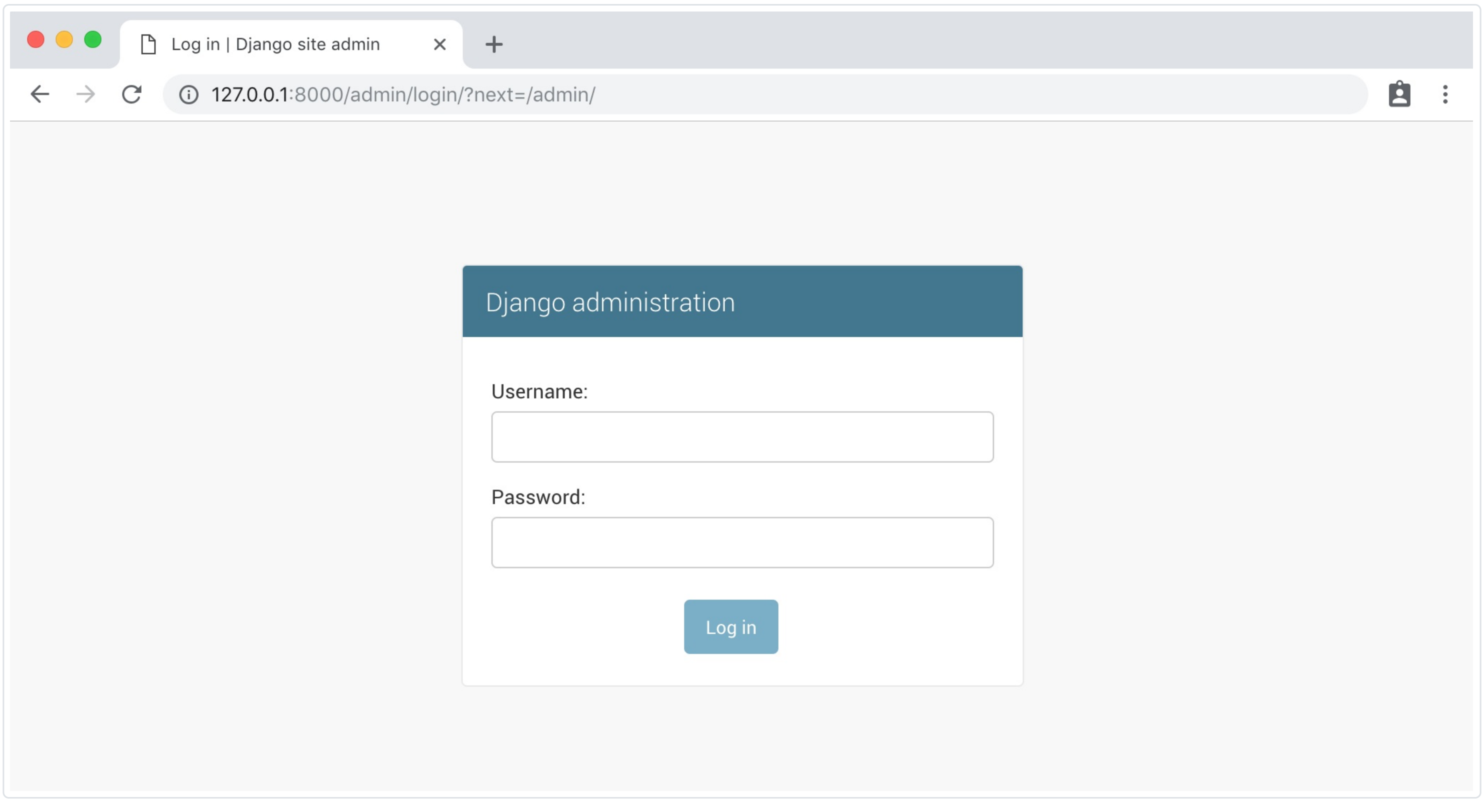
And update the admin.

```
# users/admin.py
from django.contrib import admin
from django.contrib.auth import get_user_model
from django.contrib.auth.admin import UserAdmin

from .forms import CustomUserCreationForm, CustomUserChangeForm
from .models import CustomUser

class CustomUserAdmin(UserAdmin):
    add_form = CustomUserCreationForm
    form = CustomUserChangeForm
    model = CustomUser
    list_display = ['email', 'username',]

admin.site.register(CustomUser, CustomUserAdmin)
```

Now make a migrations file, migrate the database the first time, and create a superuser so we can access the admin.

```
(code) $ python manage.py makemigrations users
(code) $ python manage.py migrate
(code) $ python manage.py createsuperuser
(code) $ python manage.py runserver
```

The local server is now running so you can visit both http://127.0.0.1:8000 for the Django welcome page and also http://127.0.0.1:8000/admin to log in with our new superuser account.


Log in | Django site admin
127.0.0.1:8000/admin/login/?next=/admin/
Django administration
Username:
Password:
Log in

Make sure to **log out** of the admin at this point. Why? Because if you don't then Django will automatically send all future URL requests to `/accounts/profile/` and future stuff won't work.

We will configure this properly later on but for now, just make sure you're logged out or stuff will break.

django-allauth

Django does not come with built-in views, urls, and templates for a sign up page. So while we can roll our own a better approach--and the current default for many professionals--is to use the django-allauth package instead.

Install it with Pipenv making sure you've used `Control+c` to stop the local server.

```
(code) $ pipenv install django-allauth~=0.42.0
```

Add it to our `INSTALLED_APPS` setting and add Django's optional sites framework which `django-allauth` uses.

```
# config/settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.sites', # new

    # 3rd party
    'allauth', # new
    'allauth.account', # new
    'allauth.socialaccount', # new

    # Local
    'users.apps.UsersConfig',
]
```

Ok, here's where the magic happens. `django-allauth` has a lengthy list of configuration options. Here's what we need.

Add this at the bottom of our `config/settings.py` file.

```
# config/settings.py
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'

AUTHENTICATION_BACKENDS = (
    # Needed to login by username in Django admin, regardless of `allauth`
    "django.contrib.auth.backends.ModelBackend",

    # `allauth` specific authentication methods, such as login by e-mail
    "allauth.account.auth_backends.AuthenticationBackend",
)

SITE_ID = 1

ACCOUNT_EMAIL_REQUIRED = True
ACCOUNT_USERNAME_REQUIRED = False
ACCOUNT_SIGNUP_PASSWORD_ENTER_TWICE = False
ACCOUNT_SESSION_REMEMBER = True
ACCOUNT_AUTHENTICATION_METHOD = 'email'
ACCOUNT_UNIQUE_EMAIL = True
```

Now run `migrate` since we've updated our `INSTALLED_APPS`.

```
(code) $ python manage.py migrate
```

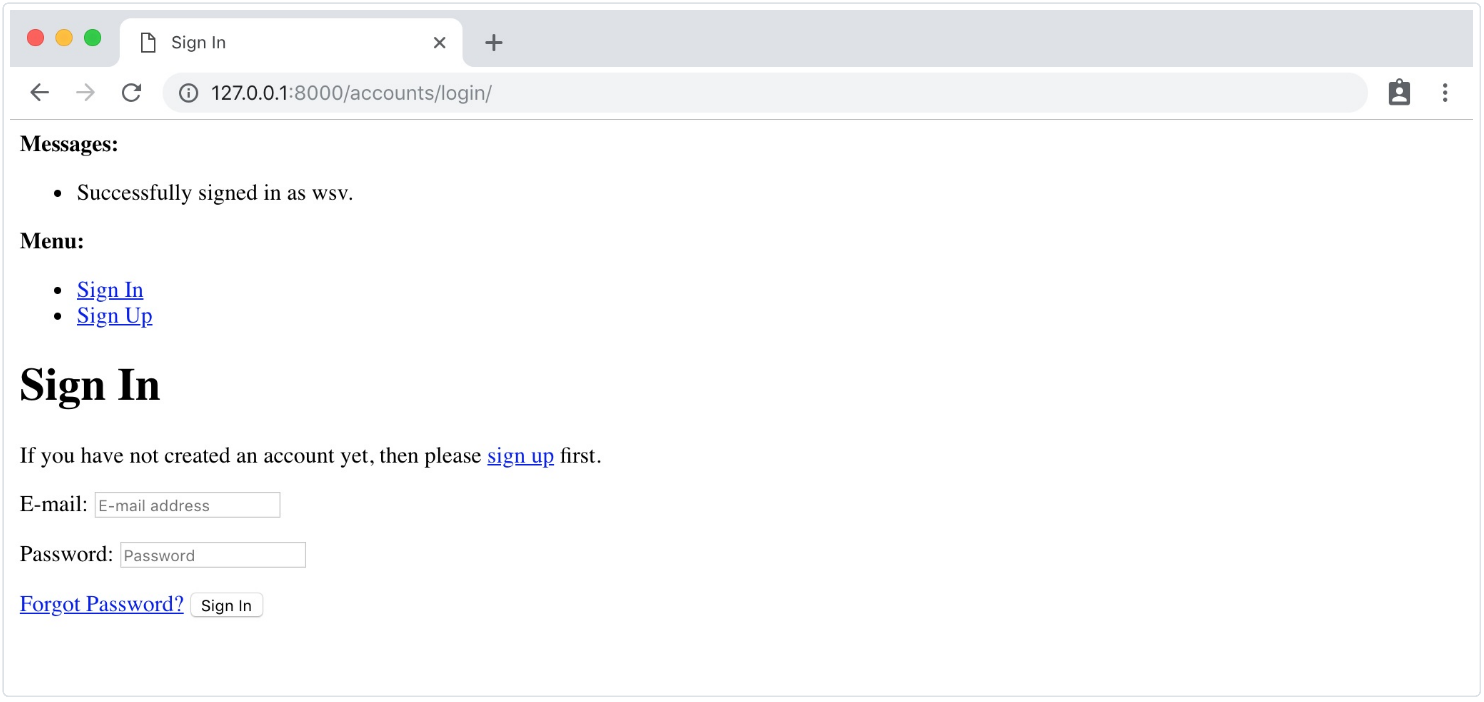
URLs

We need to tell Django *where* to put our fancy `django-allauth` config so let's use the `accounts/` path.

```
# config/urls.py
from django.contrib import admin
from django.urls import path, include # new

urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('allauth.urls')), # new
]
```

Default Log In and Sign Up Pages

If you start up the local server now with `python manage.py runserver` you can navigate to our working log in page at http://127.0.0.1:8000/accounts/login/.



Messages:

- Successfully signed in as wsv.

Menu:

- Sign In
- Sign Up

Sign In

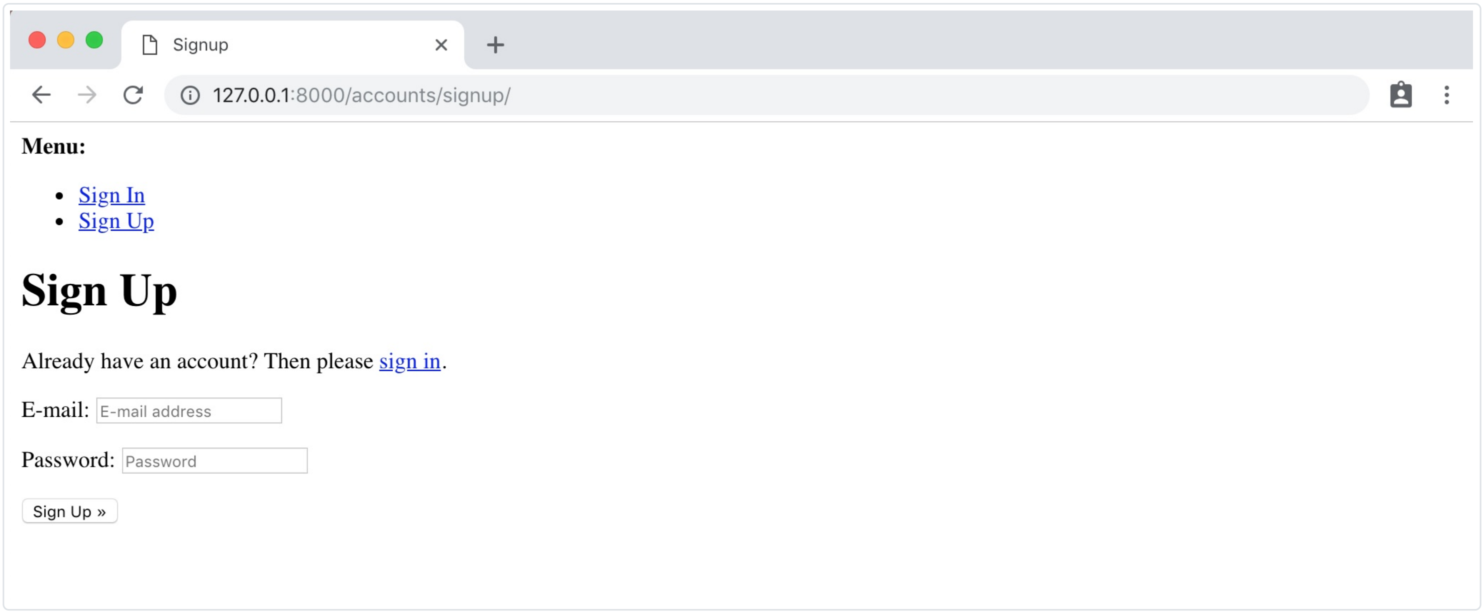
If you have not created an account yet, then please sign up first.

E-mail: E-mail address

Password: Password

Forgot Password? Sign In

Now take a look at the sign up page at http://127.0.0.1:8000/accounts/signup/.



Menu:

- Sign In
- Sign Up

Sign Up

Already have an account? Then please sign in.

E-mail: E-mail address

Password: Password

Sign Up »

Templates

Time to configure our templates so things look better. `django-allauth` uses templates in an `account` folder so we'll override its defaults there.

Create a project-level `templates` folder and then `account` within it.

```
(code) $ mkdir templates
(code) $ mkdir templates/account
```

Now tell Django to look here so update the `TEMPLATES` config within `settings.py`.

```
# config/settings.py
TEMPLATES = [
    {
        ...
        'DIRS': [os.path.join(BASE_DIR, 'templates')], # new
        ...
    }
]
```

And make our log in and sign up templates.

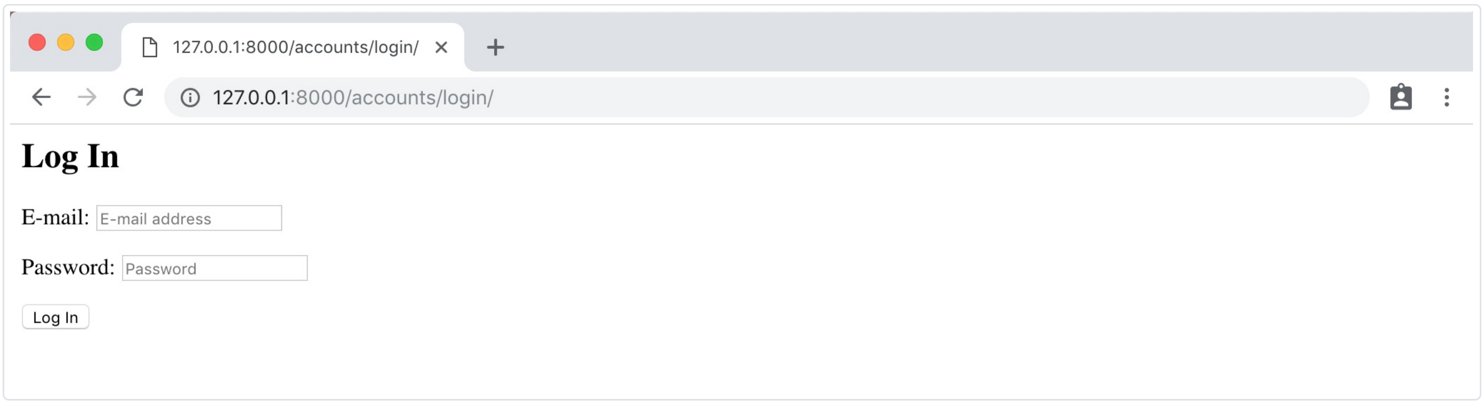
```
(code) $ touch templates/account/login.html
(code) $ touch templates/account/signup.html
```

Ok here's the code for each.

```
<!-- templates/account/login.html -->
<h2>Log In</h2>
<form method="post">
  {% raw %}{% csrf_token %}
  {{ form.as_p }}{% endraw %}
  <button type="submit">Log In</button>
</form>
```

```
<!-- templates/account/signup.html -->
<h2>Sign Up</h2>
<form method="post">
  {% raw %}{% csrf_token %}
  {{ form.as_p }}{% endraw %}
  <button type="submit">Sign Up</button>
</form>
```

Here is the new log in page.


127.0.0.1:8000/accounts/login/
127.0.0.1:8000/accounts/login/

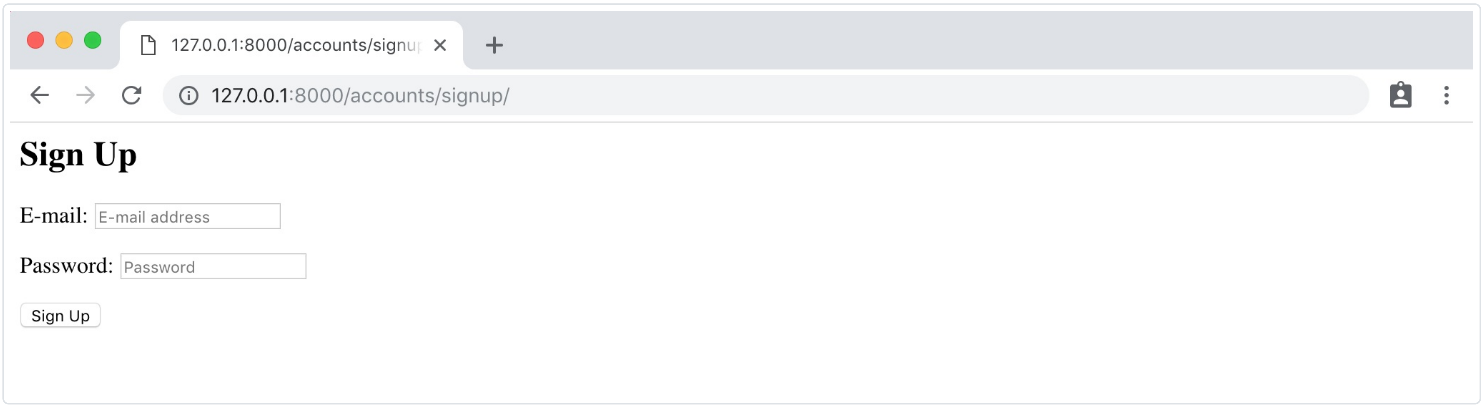
Log In

E-mail: E-mail address

Password: Password

Log In

And the new sign up page.


127.0.0.1:8000/accounts/signu
127.0.0.1:8000/accounts/signup/

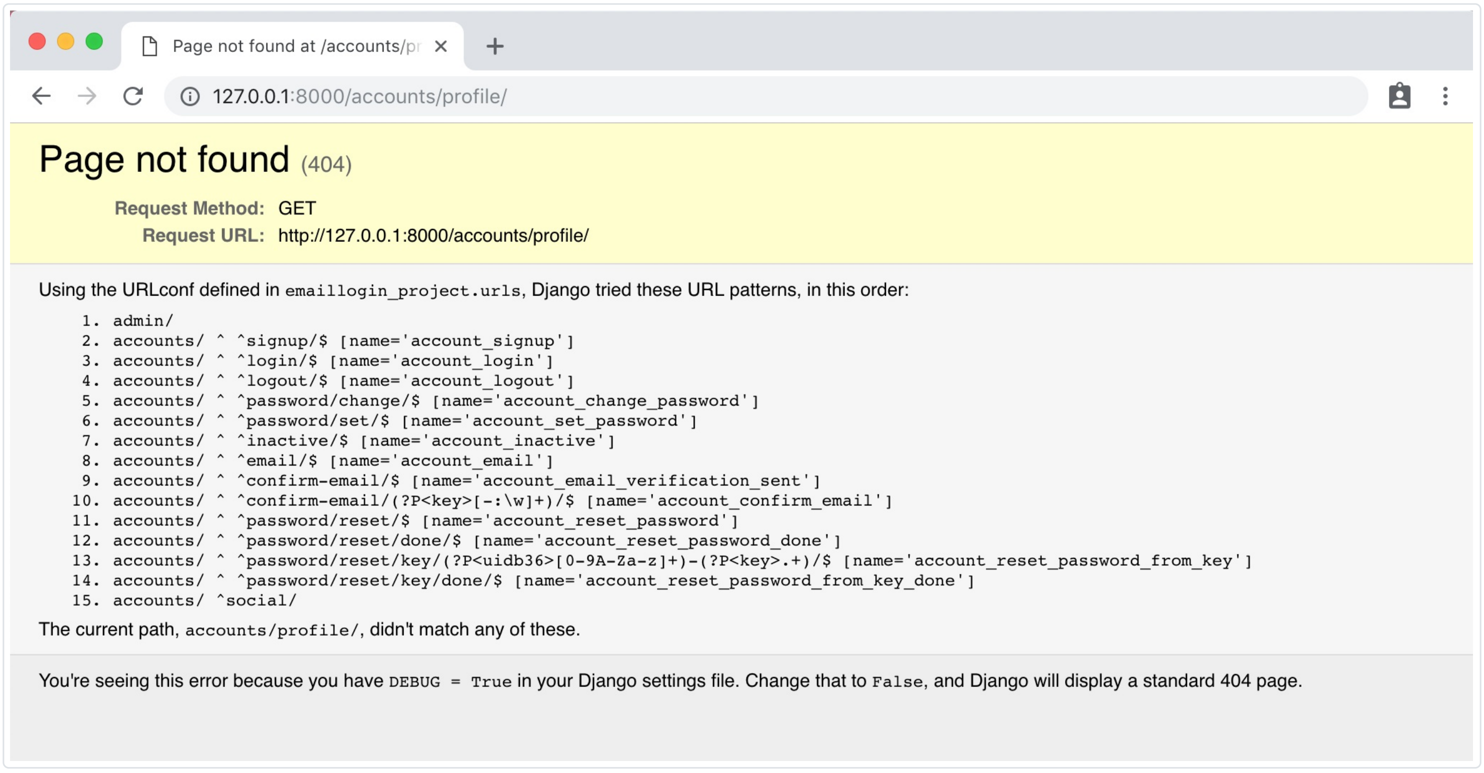
Sign Up

E-mail: E-mail address

Password: Password

Sign Up

But are we done? Haha, of course not. If you try out the forms you'll see they result in error pages. Why? We need to tell Django *where* to redirect our user after they log in and sign up.


Page not found at /accounts/pr
127.0.0.1:8000/accounts/profile/

Page not found (404)

Request Method: GET
Request URL: http://127.0.0.1:8000/accounts/profile/

Using the URLconf defined in `emaillogin_project.urls`, Django tried these URL patterns, in this order:

```
 1. admin/
 2. accounts/ ^ ^signup/$ [name='account_signup']
 3. accounts/ ^ ^login/$ [name='account_login']
 4. accounts/ ^ ^logout/$ [name='account_logout']
 5. accounts/ ^ ^password/change/$ [name='account_change_password']
 6. accounts/ ^ ^password/set/$ [name='account_set_password']
 7. accounts/ ^ ^inactive/$ [name='account_inactive']
 8. accounts/ ^ ^email/$ [name='account_email']
 9. accounts/ ^ ^confirm-email/$ [name='account_email_verification_sent']
10. accounts/ ^ ^confirm-email/(?P<key>[-:\w]+)/$ [name='account_confirm_email']
11. accounts/ ^ ^password/reset/$ [name='account_reset_password']
12. accounts/ ^ ^password/reset/done/$ [name='account_reset_password_done']
13. accounts/ ^ ^password/reset/key/(?P<uidb36>[0-9A-Za-z]+)-(?P<key>.+)/$ [name='account_reset_password_from_key']
14. accounts/ ^ ^password/reset/key/done/$ [name='account_reset_password_from_key_done']
15. accounts/ ^social/
```

The current path, `accounts/profile/`, didn't match any of these.

You're seeing this error because you have `DEBUG = True` in your Django settings file. Change that to `False`, and Django will display a standard 404 page.

Redirects

The setting for redirects is, not surprisingly, in our `settings.py` file. Here are the two lines to add:

```
# config/settings.py
LOGIN_REDIRECT_URL = 'home'
ACCOUNT_LOGOUT_REDIRECT_URL = 'home'
```

Both reference a template with the named URL of `home` so we'll need to create that homepage now.

Homepage

I like to create a dedicated `pages` app for all static pages in my Django projects. Let's blast through that quickly which means also adding a new template, view, and url. Here we go...make sure to stop the local server with `Control+c`.

```
(code) $ python manage.py startapp pages
```

Add the new `pages` app to our `INSTALLED_APPS`.

```
# config/settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.sites',

    # 3rd party
    'allauth',
    'allauth.account',
    'allauth.socialaccount',

    # Local
    'users.apps.UsersConfig',
    'pages.apps.PagesConfig', # new
]
```

Add it to our `config/urls.py` file.

```
# config/urls.py
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('allauth.urls')),
    path('', include('pages.urls')), # new
]
```

Update the views file for it.

```
# pages/views.py
from django.views.generic import TemplateView


class HomePageView(TemplateView):
    template_name = 'home.html'
```

Then make a new `pages/urls.py` file and fill it in like so:

```
# pages/urls.py
from django.urls import path

from .views import HomePageView

urlpatterns = [
    path('', HomePageView.as_view(), name='home'),
]
```

Now we can make our homepage template which will tell us if a user is logged in or not and provide helpful links.

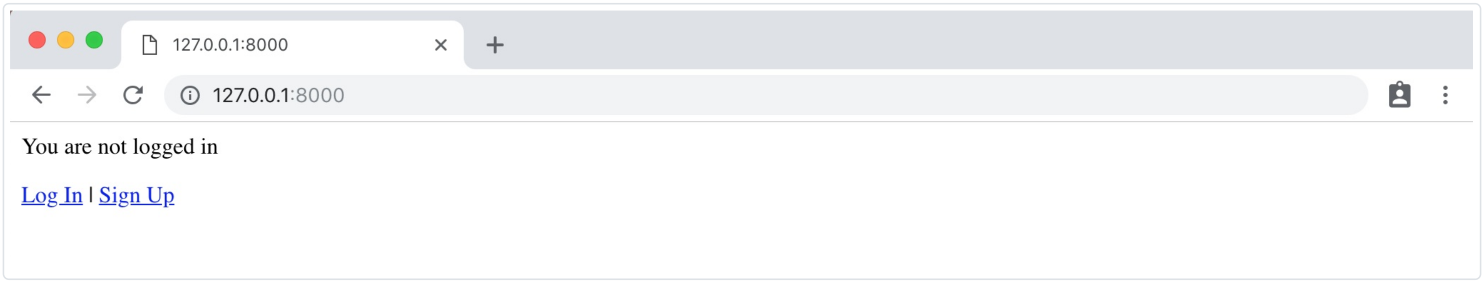
Create the `templates/home.html` file and add some basic logic for if the user is logged in or not. Be careful not to put this within the `templates/account` directory, this is just in `templates` itself.

```
<!-- templates/home.html -->
{% raw %}{% if user.is_authenticated %}
  Hi {{ user.email }}!
  <p><a href="{% url 'account_logout' %}">Log Out</a></p>
{% else %}
  <p>You are not logged in</p>
  <a href="{% url 'account_login' %}">Log In</a> |
  <a href="{% url 'account_signup' %}">Sign Up</a>
{% endif %}{% endraw %}
```

Phew! Done. Make sure you are logged out which you can confirm by heading over quickly to the Django admin http://127.0.0.1:8000/admin. The "Log out" link is in the upper right corner.

Test it all out

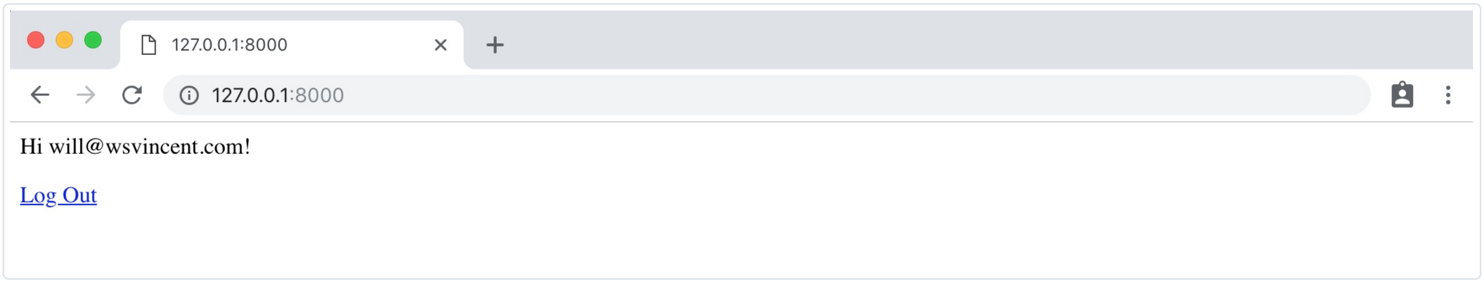
Now go to our snazzy new homepage at http://127.0.0.1:8000/.



You are not logged in

Log In | Sign Up

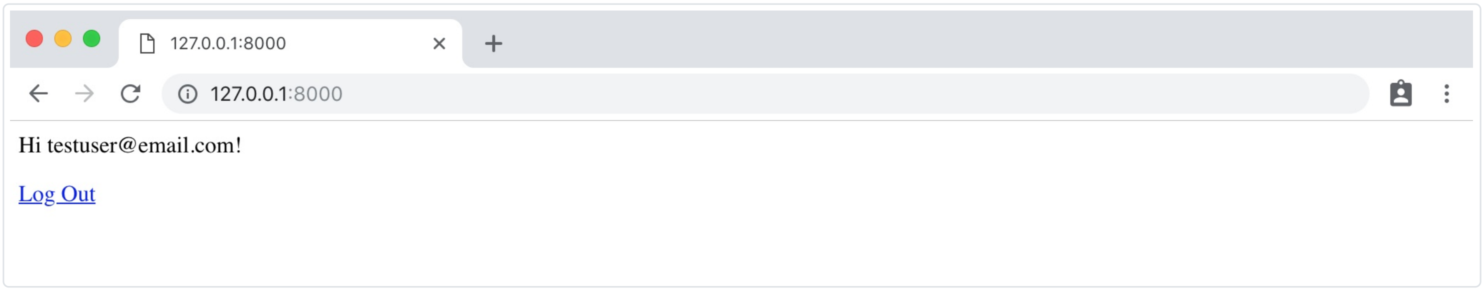
If you click on "Log in" and use your superuser account it will redirect back to the homepage. Here's what mine looks like.



Hi will@wsvincent.com!

Log Out

Now click on the "Log out" link and then try the "Sign up" link. I've created a new user account called `testuser@email.com`. Upon signing up I'm redirected back to the homepage.



Hi testuser@email.com!

Log Out

Next Steps

If you'd like to see a more full-featured Django starter project, my book Django for Professionals covers all of this in great detail as well as using Docker, PostgreSQL, and a host of more advanced Django topics.

0 Comments LearnDjango.com Disqus' Privacy Policy Login

Recommend Tweet Share Sort by Best

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

Subscribe Add Disqus to your siteAdd DisqusAdd Do Not Sell My Data DISQUS