Django Login and Logout Tutorial

In this tutorial, we'll learn how to configure login/logout functionality with Django's built-in user authentication system. This post is the first in a three-part series that also covers signup and password reset for a complete user authentication flow in your future Django projects.

This tutorial assumes you're already familiar with configuring a new Django project. If you need help, please refer to Django for Beginners, which covers the topic in more detail.

Setup

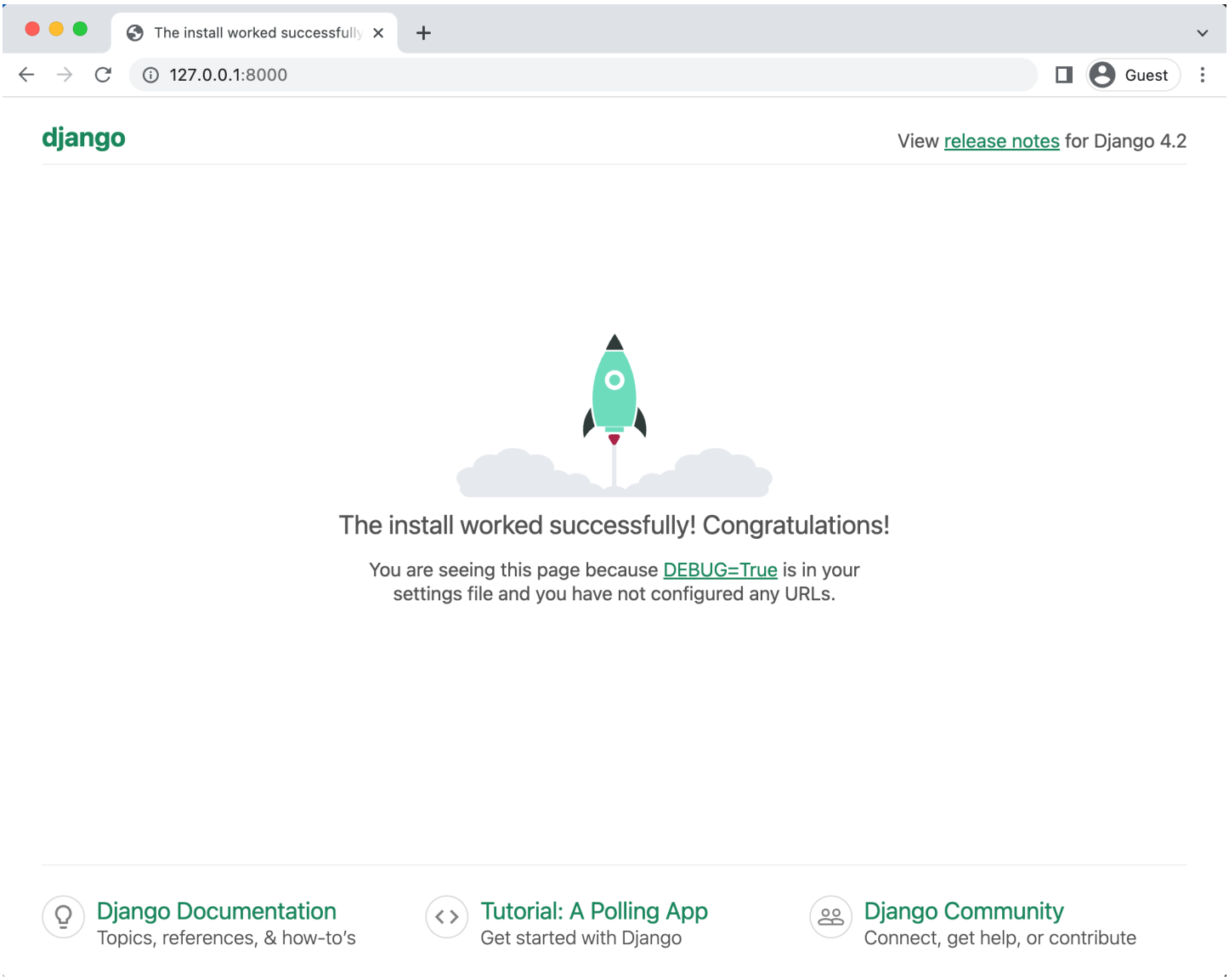
Start by creating a new Django project. This code can live anywhere on your computer. On a Mac, the Desktop is convenient, and that's where we'll put this code. We can do all of the standard configuration from the command line:

- create a new `django_auth` directory for our code on the Desktop
- create a new virtual environment called `.venv` and activate it
- install Django
- create a new Django project called `django_project`
- create a new SQLite database with `migrate`
- run the local server

Here are the commands to run:

```
# Windows
$ cd onedrive\desktop\
$ mkdir django_auth
$ cd django_auth
$ python -m venv .venv
$ .venv\Scripts\Activate.ps1
(.venv) $ python -m pip install django~=4.2.0
(.venv) $ django-admin startproject django_project .
(.venv) $ python manage.py migrate
(.venv) $ python manage.py runserver

# macOS
$ cd ~/desktop/
$ mkdir django_auth
$ cd django_auth
$ python3 -m venv .venv
$ source .venv/bin/activate
(.venv) $ python3 -m pip install django~=4.2.0
(.venv) $ django-admin startproject django_project .
(.venv) $ python manage.py migrate
(.venv) $ python manage.py runserver
```

Navigating to `http://127.0.0.1:8000`, you'll see the Django welcome screen.



The Django auth app

Django automatically installs the `auth` app when creating a new project. Look in the `django_project/settings.py` file under `INSTALLED_APPS`, and you can see `auth` is one of several built-in apps Django has installed for us.

```
# django_project/settings.py
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",  # Yoohoo!!!!
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
]
```

To use the `auth` app, we need to add it to our project-level `urls.py` file. Make sure to add `include` on the second line. I've included the `auth` app at `accounts/`, but you can use any URL pattern you want.

```
# django_project/urls.py
```

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

urlpatterns = [
    path("admin/", admin.site.urls),
    path("accounts/", include("django.contrib.auth.urls")),  # new
]
```

The `auth` app we've now included provides us with several authentication views and URLs for handling login, logout, and password management.

The URLs provided by `auth` are:

```
accounts/login/ [name='login']
accounts/logout/ [name='logout']
accounts/password_change/ [name='password_change']
accounts/password_change/done/ [name='password_change_done']
accounts/password_reset/ [name='password_reset']
accounts/password_reset/done/ [name='password_reset_done']
accounts/reset/<uidb64>/<token>/ [name='password_reset_confirm']
accounts/reset/done/ [name='password_reset_complete']
```

There are associated auth views for each URL pattern, too. That means we only need to create a *template* to use each!

Login Page

Let's make our login page! Django, by default, will look within a templates folder called `registration` for auth templates. The login template is called `login.html`.

Create a new directory called `templates` and another directory called `registration` within it.

```
(.venv) $ mkdir templates
(.venv) $ mkdir templates/registration
```

Then create a `templates/registration/login.html` file with your text editor and include the following code:

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

This code is a standard Django form using `POST` to send data and `{% csrf_token %}` tags for security concerns, namely to prevent a CSRF Attack. The form's contents are outputted between paragraph tags thanks to `{{ form.as_p }}` and then we add a "submit" button.

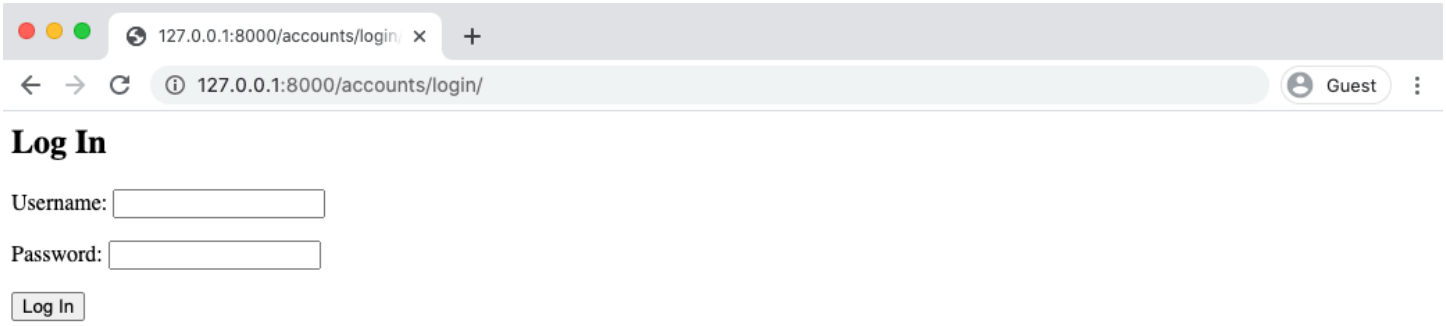
Next, update the `settings.py` file to tell Django to look for a `templates` folder at the project level. Update the `DIRS` setting within `TEMPLATES` with the following one-line change.

```
# django_project/settings.py
TEMPLATES = [
    {
        ...
        "DIRS": [BASE_DIR / "templates"],
        ...
    },
]
```

Our login functionality now works, but to improve it, we should specify *where* to redirect the user upon a successful login. In other words, where should users be sent to on the site once logged in? We use the `LOGIN_REDIRECT_URL` setting to specify this route. At the bottom of the `settings.py` file, add the following to redirect the user to the homepage.

```
# django_project/settings.py
LOGIN_REDIRECT_URL = "/"
```

We are actually done at this point! If you start the Django server again with `python manage.py runserver` and navigate to our login page at `http://127.0.0.1:8000/accounts/login/`, you'll see the following.

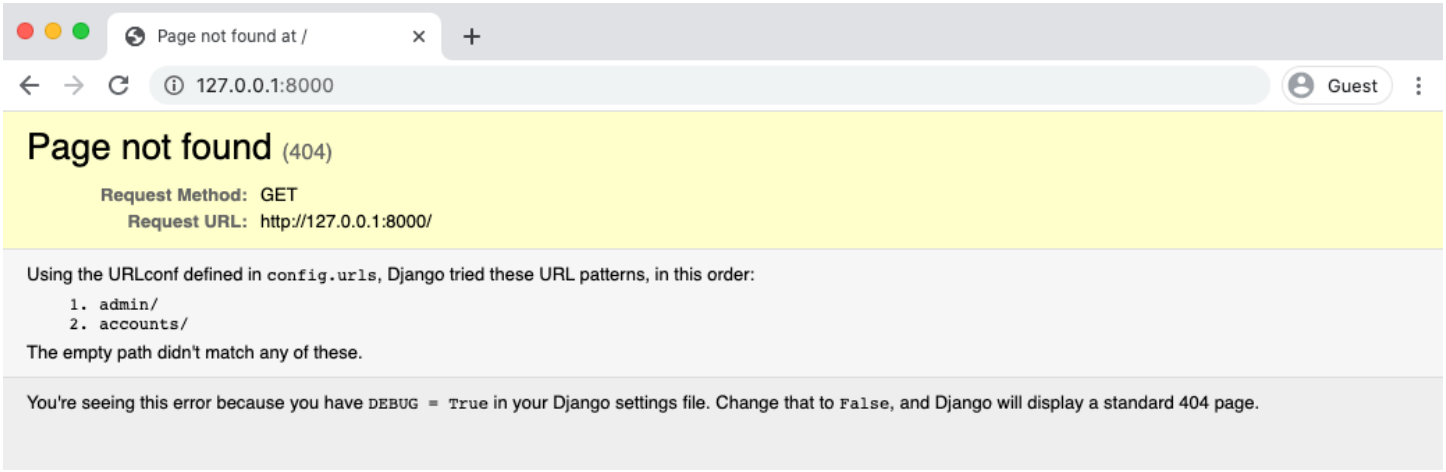


Create users

But there's one missing piece: **we still need to create users**. Let's quickly make a superuser account from the command line. Quit the server with `Control+c` and then run the command `python manage.py createsuperuser`. Answer the prompts and note that your password will not appear on the screen when typing for security reasons.

```
(.venv) > python manage.py createsuperuser
Username (leave blank to use 'wsv'):
Email address: will@wsvincent.com
Password:
Password (again):
Superuser created successfully.
```

Now start the server again with `python manage.py runserver` and refresh the page at `http://127.0.0.1:8000/accounts/login/`. Enter the login info for your just-created user.



Our login worked because it redirected us to the homepage, but we still need to create that homepage, so we see the error *Page not found*. Let's fix that!

Create a homepage

We want a simple homepage displaying one message to logged-out users and another to logged-in users. Create two new files with your text editor: `templates/base.html` and `templates/home.html.` Note that these files exist within the `templates` folder but *not* within `templates/registration/`, where Django auth looks by default for user auth templates.

Add the following code to each:

```
<!-- templates/base.html -->
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>{% block title %}Django Auth Tutorial{% endblock %}</title>
</head>
<body>
  <main>
    {% block content %}
    {% endblock %}
  </main>
</body>
</html>
```

```
<!-- templates/home.html -->
{% extends "base.html" %}

{% block title %}Home{% endblock %}

{% block content %}
{% if user.is_authenticated %}
  Hi {{ user.username }}!
{% else %}
  <p>You are not logged in</p>
  <a href="{% url 'login' %}">Log In</a>
{% endif %}
```

```
{% endblock %}
```

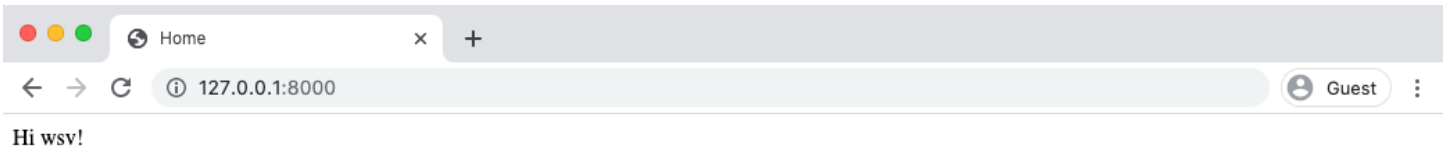
While we're at it, we can update `login.html` too to extend our new `base.html` file:

```
<!-- templates/registration/login.html -->
{% extends "base.html" %}

{% block title %}Login{% endblock %}

{% block content %}
<h2>Log In</h2>
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Log In</button>
</form>
{% endblock %}
```

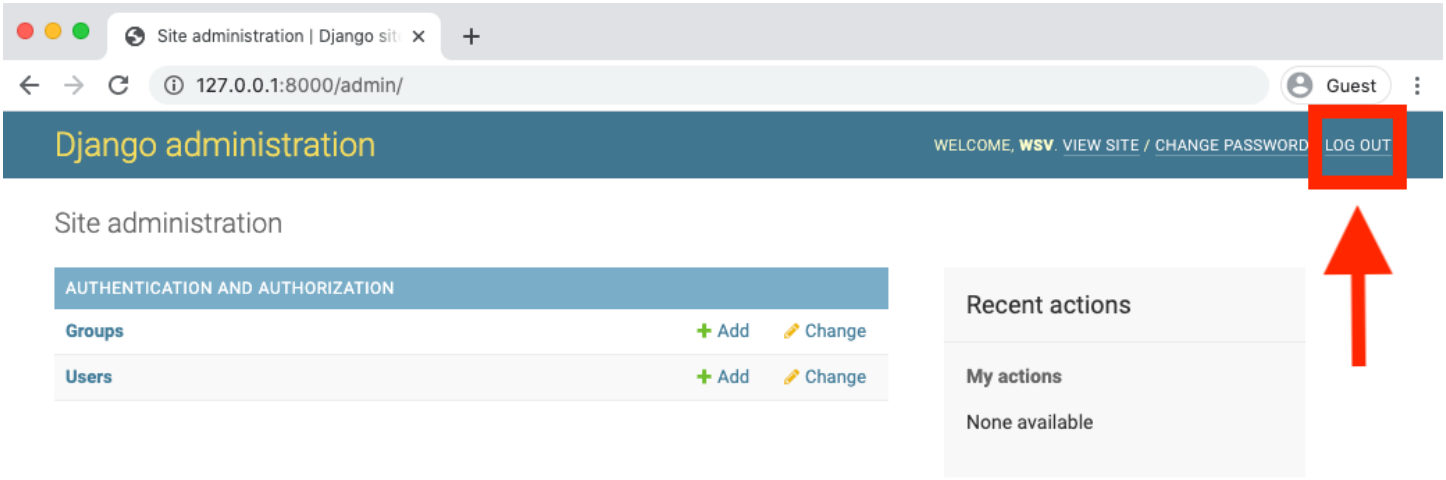
Now, update the `django_project/urls.py` file so we can display the homepage. Usually, I prefer to create a dedicated `pages` app for this purpose. Still, we can do it for simplicity within our existing `django_project/urls.py` file: import `TemplateView` on the third line and then add a URL pattern for it at the path `""`.

```
# django_project/urls.py
from django.contrib import admin
from django.urls import path, include
from django.views.generic.base import TemplateView  # new

urlpatterns = [
    path("admin/", admin.site.urls),
    path("accounts/", include("django.contrib.auth.urls")),
    path("", TemplateView.as_view(template_name="home.html"), name="home"),  # new
]
```

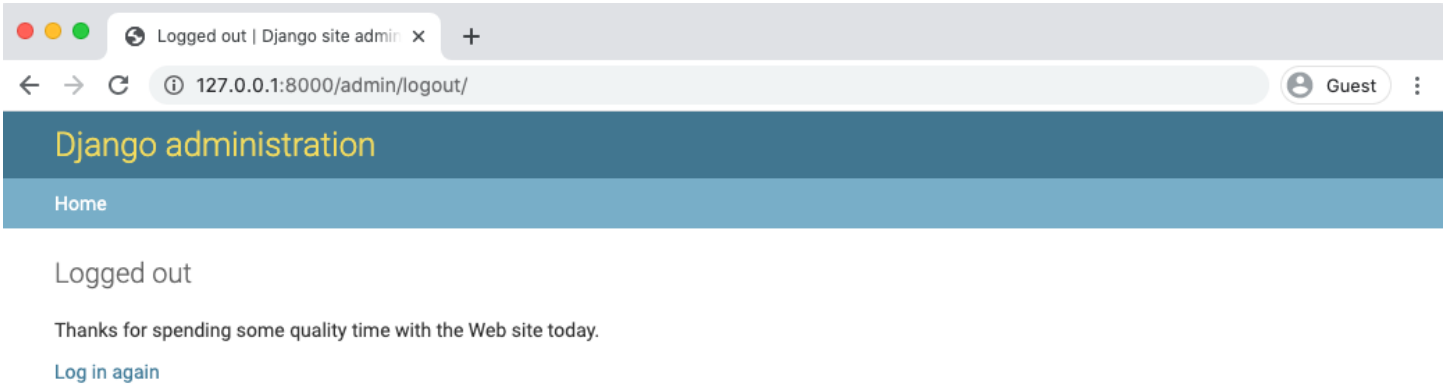
And we're done. If you start the Django server again with `python manage.py runserver` and navigate to the homepage at `http://127.0.0.1:8000/` you'll see the following:



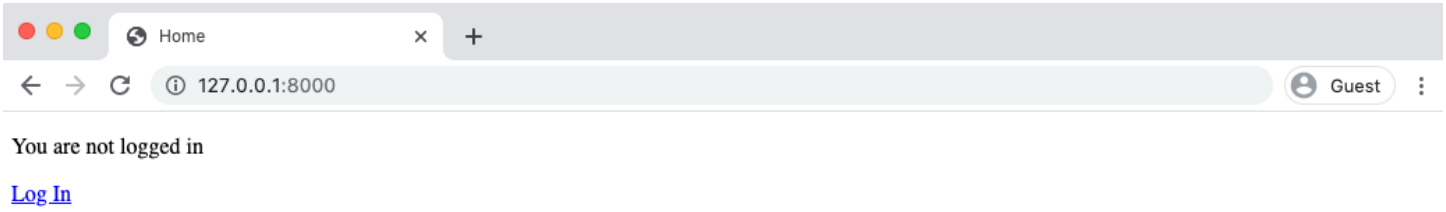
It worked! But how do we log out? The only option currently is to go into the admin panel at `http://127.0.0.1:8000/admin/` and click the "Logout" link in the upper right corner.



The "Logout" link will log us out, as seen by the redirect page:



If you go to the homepage again at `http://127.0.0.1:8000/` and refresh the page, it is visible that we are logged out.



Logout link

Let's add a logout link to our page so users can easily toggle back and forth between the two states. Fortunately, the Django `auth` app already provides a built-in URL and view. And if you think about it, we don't need to display anything on logout, so there's no need for a template. After a successful "logout" request, we are all really redirected to another page.

So let's first add a link to the built-in `logout` URL in our `home.html` file:

```
<!-- templates/home.html-->
```

```
{% extends "base.html" %}

{% block title %}Home{% endblock %}

{% block content %}
{% if user.is_authenticated %}
  Hi {{ user.username }}!
  <p><a href="{% url 'logout' %}">Log Out</a></p>
{% else %}
  <p>You are not logged in</p>
  <a href="{% url 'login' %}">Log In</a>
{% endif %}
{% endblock %}
```

Then update `settings.py` with our redirect link, `LOGOUT_REDIRECT_URL`. Add it right next to our login redirect so the bottom of the `settings.py` file should look as follows:

```
# django_project/settings.py
LOGIN_REDIRECT_URL = "/"
LOGOUT_REDIRECT_URL = "/"  # new
```

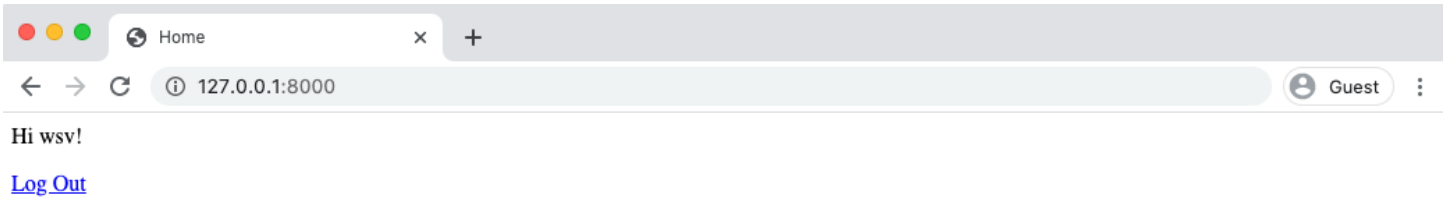
Now that we have a homepage view, we should use that instead of our current hardcoded approach. What's the URL name of our homepage? It's `home`, which we named in our `django_project/urls.py` file:

```
# django_project/urls.py
    ...
    path("", TemplateView.as_view(template_name="home.html"), name="home"),
    ...
```

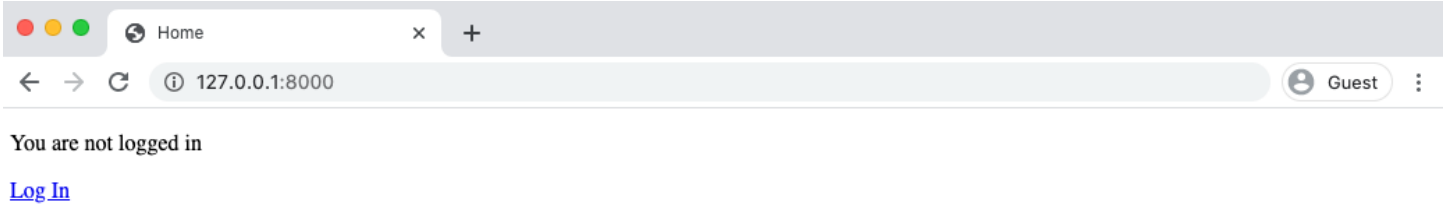
So we can replace `"/"` with `home` at the bottom of the `settings.py` file:

```
# django_project/settings.py
LOGIN_REDIRECT_URL = "home"
LOGOUT_REDIRECT_URL = "home"
```

If you revisit the homepage and log in, you'll be redirected to the new homepage with a "logout" link for logged-in users.



Clicking it takes you to the homepage with a "login" link.



Conclusion

With very little code, we have a robust login and logout authentication system. It probably feels like magic since the `auth` app did much of the heavy lifting for us. However, the benefit of Django's "batteries-included" approach is that it provides a lot of functionality out-of-the-box while leaving room for plenty of customization if desired.