



Published in Python in Plain English

You have **1** free member-only story left this month. [Sign up for Medium and get an extra one](#)



Esther Vaati

[Follow](#)Aug 29 · 8 min read · ✨ · [Listen](#)

Save



How to Build Your Own Login and Registration System in Django

User Authentication in Django



Photo by [Desola Lanre-Ologun](#) on [Unsplash](#)

Django comes with the necessary features to set up a complete user authentication system. In this tutorial, we'll cover the most critical components of a user authentication system in Django, namely:

- Registration
- Login
- Logout

By the end of this tutorial, you should be able to

- Create a Django project where users can register, login, and logout.

Open in app ↗

Sign up

Sign In



Prerequisites

This tutorial assumes you have a good knowledge of Django and you have Python3 installed in your operating system.

Create a Django Project

It is recommended to set up a virtual environment to ensure that project dependencies are isolated from system packages . Virtual environments also makes it easier to install the correct versions of modules.

Start by creating a directory to house your project files.

```
mkdir Django_Login
```

cd to the directory you have created above start by creating a virtual environment with the python venv module.

```
cd Django_Login  
python -m venv my_env
```

The `venv` module provides support for creating lightweight “virtual environments” with their own site directories, optionally isolated from system site directories. Each virtual environment has its own Python binary (which matches the version of the binary that was used to create this environment) and can have its own independent set of installed Python packages in its site directories.

Python Virtual Environments Explained

How to use Venv and Anaconda in Your Python and Data science projects

python.plainenglish.io



37



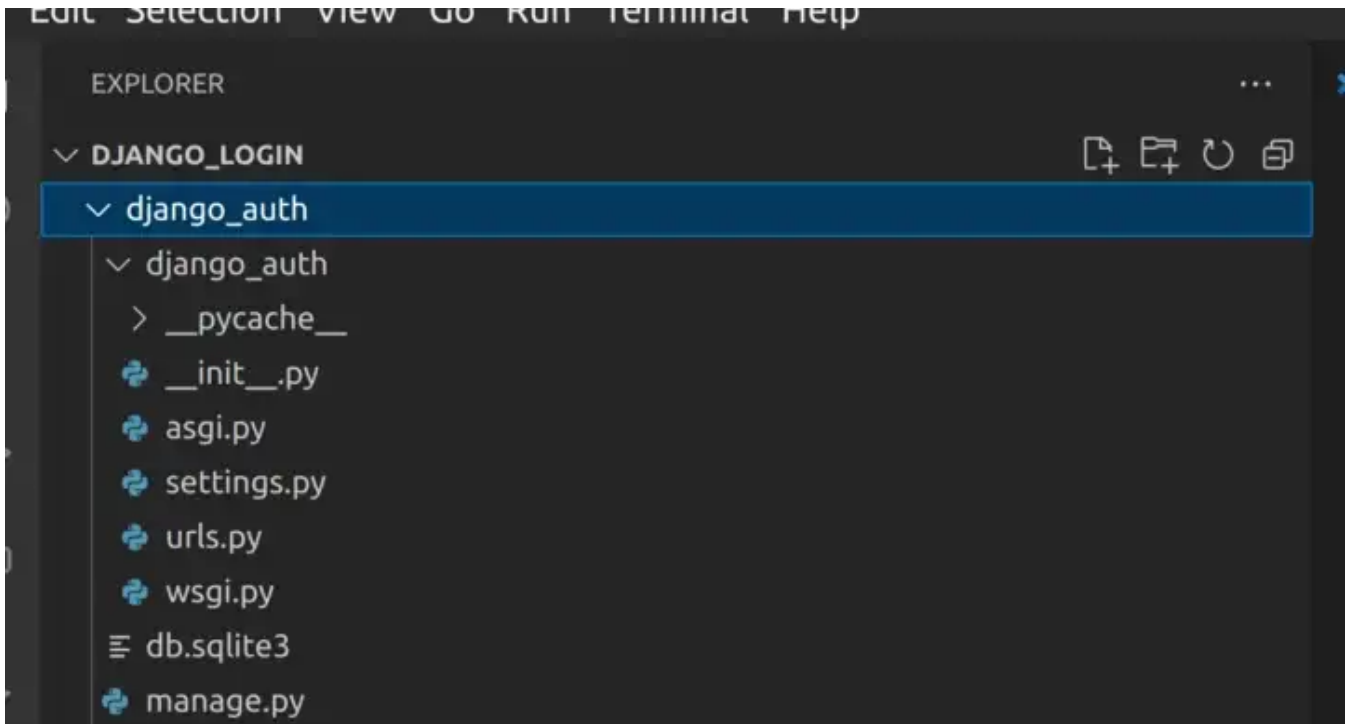
Activate the virtual environment and install Django.

```
source my_env/bin/activate  
pip install django
```

Create a Django project called `django_auth`

```
django-admin startproject django_auth
```

Your project directory now looks like this:



User Authentication in Django

Django comes complete with an authentication system so you don't have to create an authentication one from scratch.

The Django authentication system is provided by `django.contrib.auth` and consists of the following

Users

Permissions: Binary (yes/no) flags designating whether a user may perform a certain task.

Groups: A generic way of applying labels and permissions to more than one user.

A configurable password hashing system

Forms and view tools for logging in users, or restricting content

A pluggable backend system

`django.contrib.auth` provides all the logic needed to successfully register, login and logout users and restrict content to unauthorized users.

`django.contrib.auth` is already present in your Django project. If you open the `settings.py` file, you will find the configurations namely `django.contrib.auth` and `django.contrib.contenttypes` in the list of `INSTALLED_APPS`, as shown below.

```
1  INSTALLED_APPS = [  
2      'django.contrib.admin',  
3      'django.contrib.auth',  
4      'django.contrib.contenttypes',  
5      'django.contrib.sessions',  
6      'django.contrib.messages',  
7      'django.contrib.staticfiles',  
8  ]
```

settings.py hosted with ❤ by GitHub

[view raw](#)

You can also find auth and sessions middleware configurations in the settings.py file

```
1  MIDDLEWARE = [  
2      'django.middleware.security.SecurityMiddleware',  
3      'django.contrib.sessions.middleware.SessionMiddleware',  
4      'django.middleware.common.CommonMiddleware',  
5      'django.middleware.csrf.CsrfViewMiddleware',  
6      'django.contrib.auth.middleware.AuthenticationMiddleware',  
7      'django.contrib.messages.middleware.MessageMiddleware',  
8      'django.middleware.clickjacking.XFrameOptionsMiddleware',  
9  ]
```

settings.py hosted with ❤ by GitHub

[view raw](#)

Authentication Database Tables

Running the migrate command will create the necessary tables for the user authentication system. Lets run the command

```
python manage.py migrate
```

The command will create actual tables in our database, and you should see something like this:

```
1  Operations to perform:
2  Apply all migrations: admin, auth, contenttypes, sessions
3  Running migrations:
4  Applying contenttypes.0001_initial... OK
5  Applying auth.0001_initial... OK
6  Applying admin.0001_initial... OK
7  Applying admin.0002_logentry_remove_auto_add... OK
8  Applying admin.0003_logentry_add_action_flag_choices... OK
9  Applying contenttypes.0002_remove_content_type_name... OK
10 Applying auth.0002_alter_permission_name_max_length... OK
11 Applying auth.0003_alter_user_email_max_length... OK
12 Applying auth.0004_alter_user_username_opts... OK
13 Applying auth.0005_alter_user_last_login_null... OK
14 Applying auth.0006_require_contenttypes_0002... OK
15 Applying auth.0007_alter_validators_add_error_messages... OK
16 Applying auth.0008_alter_user_username_max_length... OK
17 Applying auth.0009_alter_user_last_name_max_length... OK
18 Applying auth.0010_alter_group_name_max_length... OK
19 Applying auth.0011_update_proxy_permissions... OK
20 Applying auth.0012_alter_user_first_name_max_length... OK
21 Applying sessions.0001_initial... OK
```

migrate.sh hosted with ❤ by GitHub

[view raw](#)

You can also view the tables and confirm that they have been created:

```
1  python3.8 manage.py dbshell
2  SQLite version 3.11.0 2016-02-15 17:29:24
3  Enter ".help" for usage hints.
4  sqlite> .tables
5  auth_group                auth_user_user_permissions
6  auth_group_permissions    django_admin_log
7  auth_permission           django_content_type
8  auth_user                 django_migrations
9  auth_user_groups          django_session
10 sqlite>
```

db.sh hosted with ❤ by GitHub

[view raw](#)

As you can see above, we have the following tables:

- auth_group -
- auth_group_permissions
- auth_user

- `auth_user_groups-`

Create a superuser

Let's create a superuser so that we can have access to the Django admin site:

```
python manage.py createsuperuser
```

Run the development server.

```
python manage.py runserver
```

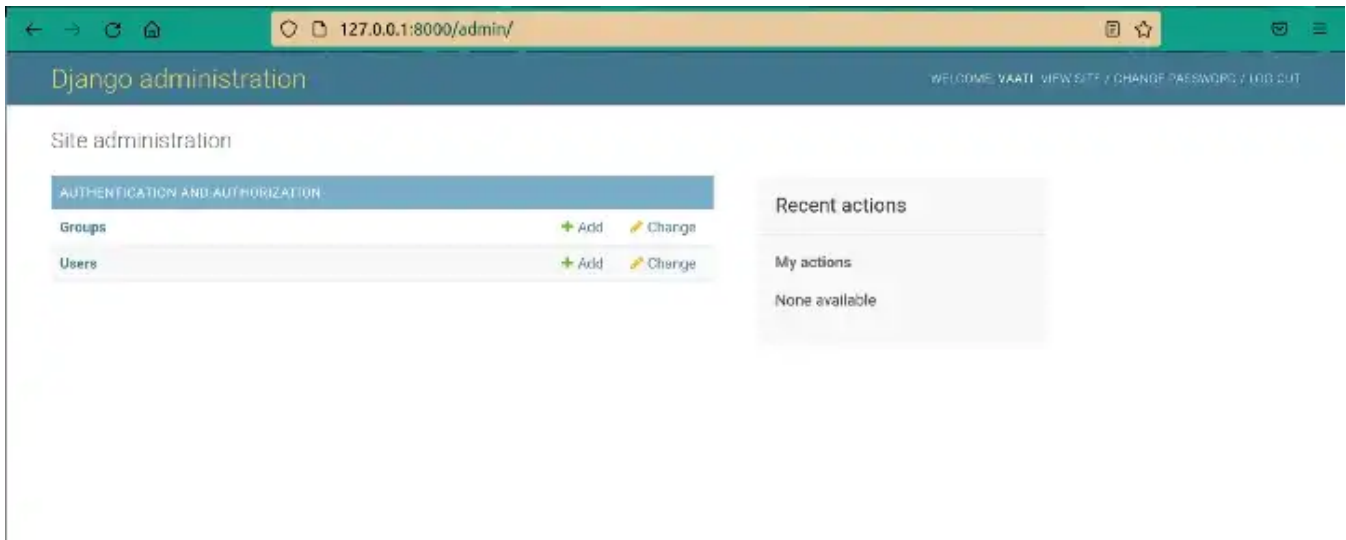
You should see something like this:

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
August 25, 2022 - 08:10:00
Django version 4.0.6, using settings 'django_auth.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

The application should start at <http://127.0.0.1:8000/>. If you navigate to <http://127.0.0.1:8000/admin>, you can be able to login with the superuser details you created above.

Once you login, the admin site looks like this



As you can see above, you can create users from the admin site, but that's not what we want. We want to provide an interface where users can register, log in, and log out independently.

Authentication Views

Since Django has already provided all the login needed to register, login and logout, we need to create pages where users can do their own authentication.

To do that , Django provides the `django.contrib.auth.urls` ,include them in the root `urls.py` file as shown below.

```
1 #django_auth/urls.py
2 from django.contrib import admin
3 from django.urls import path,include
4 urlpatterns = [
5     path('admin/', admin.site.urls),
6     path('accounts/', include('django.contrib.auth.urls')),
7 ]
```

auth_urls.py hosted with ♥ by GitHub

[view raw](#)

The URLs provided by Django 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']
```

Create users app

Lets create a users app which will handle all the stuff to do with authentication.

Create a Django app called users with the `startapp` command:

```
python manage.py startapp users
```

The command above will create a directory called `users` in the `django_auth` project.

Your files should now look like this.

```
1  django_auth/
2      manage.py
3      django_auth/
4          __init__.py
5          settings.py
6          urls.py
7          asgi.py
8          wsgi.py
9      users/
10         __init__.py
11         admin.py
12         apps.py
13         migrations/
14         models.py
15         tests.py
16         views.py
```

dir.sh hosted with ♥ by GitHub

[view raw](#)

Since Django already comes with the User model, we will not create models for this app.

Open `settings.py` and add the users app in `INSTALLED_APPS` .

```
1  INSTALLED_APPS = [  
2      'django.contrib.admin',  
3      'django.contrib.auth',  
4      'django.contrib.contenttypes',  
5      'django.contrib.sessions',  
6      'django.contrib.messages',  
7      'django.contrib.staticfiles',  
8      'users',  
9  ]
```

users_app.py hosted with ♥ by GitHub

[view raw](#)

Create a Home Page

A home page will be the first page a user sees when they open our application.

Whenever Django wants to render a page, it will automatically look for it in the templates directory of the app. The best practice is to place the HTML page inside a directory with the same name as the app.

Create a templates folder in the users app. Inside the templates you have just created, create another folder `users` . Next create a file `home.html` in the inner most `users` folder. Your structure of the users app should now look like this:

```
1  users/  
2      __init__.py  
3      admin.py  
4      apps.py  
5      migrations/  
6      models.py  
7      templates/  
8          users/  
9              home.html  
10     tests.py  
11     urls.py  
12     views.py
```

dir.sh hosted with ♥ by GitHub

[view raw](#)

Add the following code to `home.html` :

```
1  <html>
2    <head>
3      <style>
4        body {
5          color: pink;
6          text-align: center;
7        }
8
9      </style>
10   </head>
11   <body>
12     <h1>Hello</h1>
13
14     <p> Welcome to the Our site</p>
15     <div>
16       <a href="{% url 'signup' %}">Sign Up</a>
17
18       <a href="{% url 'login' %}">Login</a>
19     </div>
20   </body>
21 </html>
```

home.html hosted with ❤ by GitHub

[view raw](#)

Dont worry about the url's, they dont work for now.

Let's now render the home page. Django provides the `render()` function, which loads the template and provides the contexts when they are passed as arguments. Open `users/views.py` and write the following view, which renders the home page.

```
1  from django.shortcuts import render
2
3  # Create your views here.
4
5  def home(request):
6      return render(request, "users/home.html")
```

users_views.py hosted with ❤ by GitHub

[view raw](#)

Map the View to a URL

Create a file called `urls.py` in the `users` app directory and add the URL for the home page. At the top of the `urls.py` file, import the `path` object from `django.urls` and the `home` view function from `views.py`:

```
1 # users/urls.py
2 from django.urls import path
3 from .views import home, SignUp
4
5 urlpatterns = [
6     path('', home, name = "home"),
7 ]
```

urls.py hosted with ♥ by GitHub

[view raw](#)

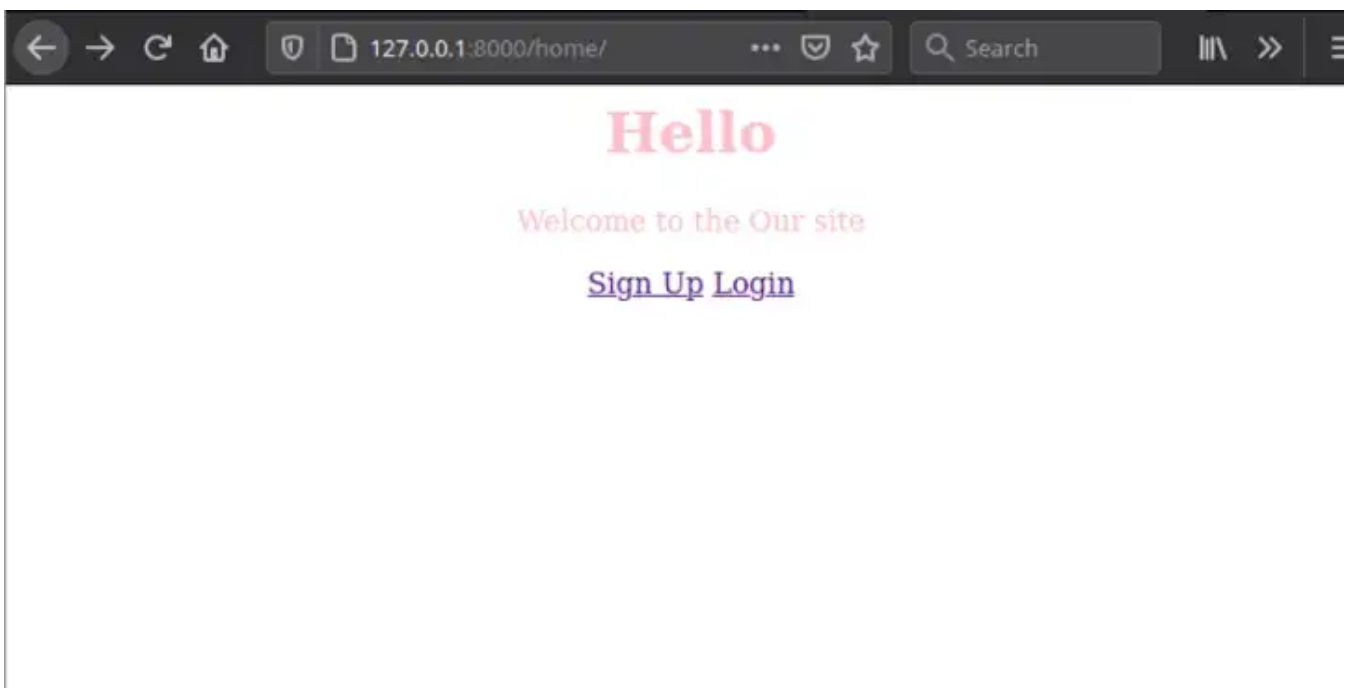
Don't forget to add the users app URLs to the project's URLs. The `django_auth/urls.py` file should now look like this:

```
1 #django_auth/urls.py
2
3 from django.contrib import admin
4 from django.urls import path, include
5
6 urlpatterns = [
7     path('admin/', admin.site.urls),
8     path('accounts/', include('django.contrib.auth.urls')),
9     path('home/', include('users.urls')),
10
11 ]
```

urls.py hosted with ♥ by GitHub

[view raw](#)

Now when you run the server and navigate to 127.0.0.1:8000/home/, you should see the following page:



Authentication Views

Django provides a list of authentication view classes which include all the logic for performing their unique purposes. These view are:

- class LoginView
- class LogoutView
- *class* PasswordChangeView
- class PasswordChangeDoneView
- class PasswordResetView

Since these views are already implemented, we don't need to write the logic from scratch, rather we need to provide the template for each view.

Create Login Page

By default, the <http://127.0.0.1:8000/accounts/login/> url will try to render the login template . By default, the name of the template for logging users should be **registration/login.html** .

Django will look for the login template in the registration folder in the templates directory. Let's create a folder in the templates directory of the users app called `registration` and add the `login.html` file:

```
1  users/
2      __init__.py
3      admin.py
4      apps.py
5      migrations/
6      models.py
7      templates/
8          users/
9              home.html
10             registration/
11                 login.html
12     tests.py
13     urls.py
14     views.py
```

dir.sh hosted with ♥ by GitHub

[view raw](#)

Add the following code the in the `login.html` file

```
1  <!-- registration/login.html -->
2
3  {% block title %}Login{% endblock %}
4
5  {% block content %}
6      <h2>Login</h2>
7
8      <form method="post">
9          {% csrf_token %}
10         <table>
11             {{ form.as_p }}
12         <tr>
13             <td>&nbsp;</td>
14             <td><input type="submit" value="Submit"></td>
15         </tr>
16     </table>
17 </form>
18 {% endblock %}
```

login.html hosted with ♥ by GitHub

[view raw](#)

Here we add a `csrf_token` to our form. Django uses CSRF tokens to protect forms from malicious users. It does this by adding a secret token inside the POST method when the form is rendered. `{{ form.as_p }}` renders the form as a series of `<p>` tags, with each `<p>` containing one field.

Now, you can now be able to log in. If you navigate to <http://127.0.0.1:8000/accounts/login/>, you should see the following page:

Login

Login

Username:

Password:

Submit

Login page

The login form contains 2 fields username and password. When a user submits the login form with the proper credentials, the user is redirected to the URL specified in the next. If the user login is not successful, the login form is redisplayed.

Let's try and log in with the wrong credential, you should see the errors below:

Login

Login

- Please enter a correct username and password. Note that both fields may be case-sensitive.

Username:

Password:

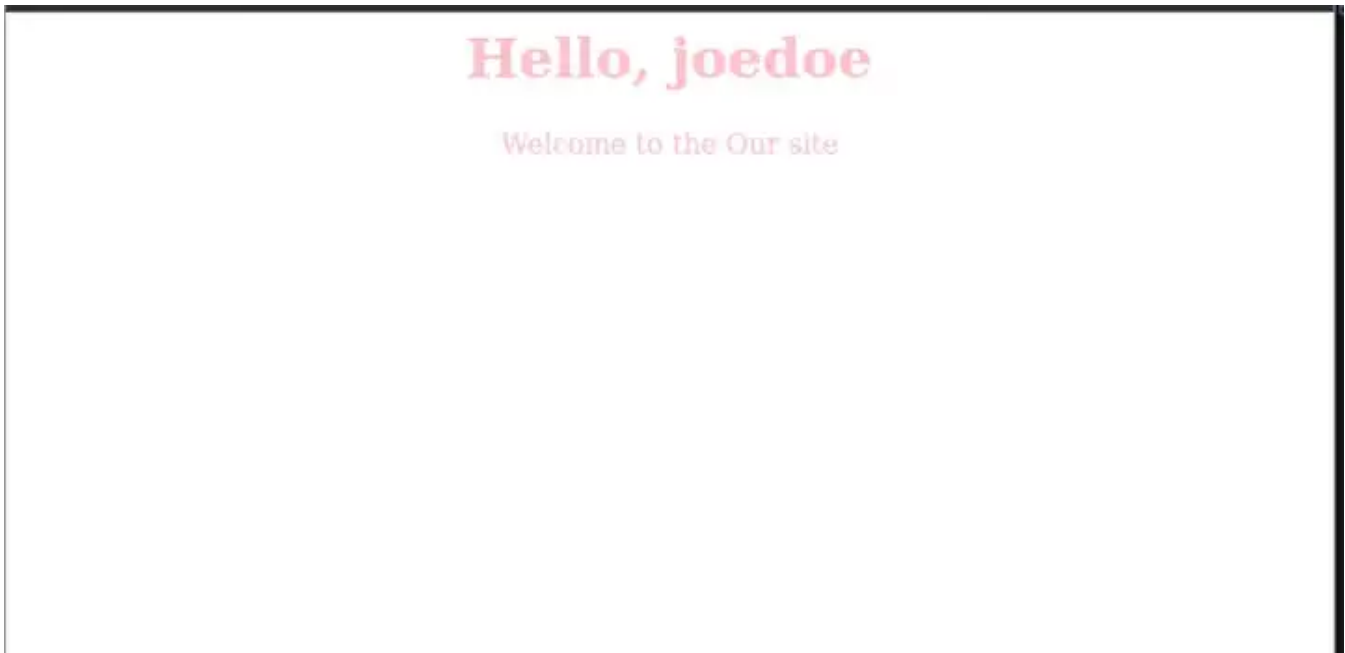
Submit

To provide a seamless user experience, we need to redirect users to the home page after successful login. To do that, add the following in `settings.py`.

```
# users will be redirected to the home page after login
```

```
LOGIN_REDIRECT_URL = "home"
```

After successful login., you should see the following welcome page.



Home Page

Create a Logout Page

To log out users, we will add a link that will log out users if they click on it. We will also add a redirect URL in `settings.py`, which will redirect users back to the home page once they log out. Open the `settings.py` file and add the following line of code

```
LOGOUT_REDIRECT_URL = "login"
```

Open `home.html` and update it as shown below:


```
1  <body>
2
3  <h1>Hello, {{ user.username|default:'Guest' }}</h1>
4
5  <p> Welcome to the Our site</p>
6  <div>
7      {% if user.is_authenticated %}
8          <a href="{% url 'logout' %}">Logout</a>
9
10         {% else %}
11             <a href="{% url 'login' %}">Login</a>
12         {% endif %}
13 </div>
14
15 </body>
```

home.html hosted with ♥ by [GitHub](#)

[view raw](#)

Here we check if a user is authenticated, and show the logout link if True, and if False, we show the login link.



Hello, Guest

Welcome to the Our site

[Login](#)

Create a User Registration Page

Up to now, we have been relying on the superuser to perform login and logout functions. We are now able to log in and log out successfully. Let's see how users can register on their own without having to use the Django admin.

The easiest way to do this is to use the `UserCreationForm` and the `CreateView` class-based view, which Django provides. The `UserCreationForm` is a `ModelForm` for

creating a new user and it generates the necessary fields i.e., username and password.

Open `user/views.py` and add the `SignUp` view class:

```
1  #users/views.py
2
3  from django.shortcuts import render
4  from django.urls import reverse_lazy
5  from django.contrib.auth.forms import UserCreationForm
6  from django.views.generic.edit import CreateView
7
8  # Create your views here.
9
10 def home(request):
11     return render(request, "users/home.html")
12
13
14
15 class SignUp(CreateView):
16     form_class = UserCreationForm
17     success_url = reverse_lazy("login")
18     template_name = "registration/signup.html"
```

`view.py` hosted with ❤ by GitHub

[view raw](#)

Next, create the `templates/registration/signup.html` page and add the following:

```
1  <!--/templates/register.html-->
2
3  {% block content %}
4  <h2>Register</h2>
5
6  <form method="post">
7      {% csrf_token %}
8      {{form}}
9      <input type="submit" value="Register">
10 </form>
11
12 <a href="{% url 'login' %}">Back to login</a>
13 <a href="{% url 'home' %}">Back to Home</a>
14 {% endblock %}
```

`signup.html` hosted with ❤ by GitHub

[view raw](#)

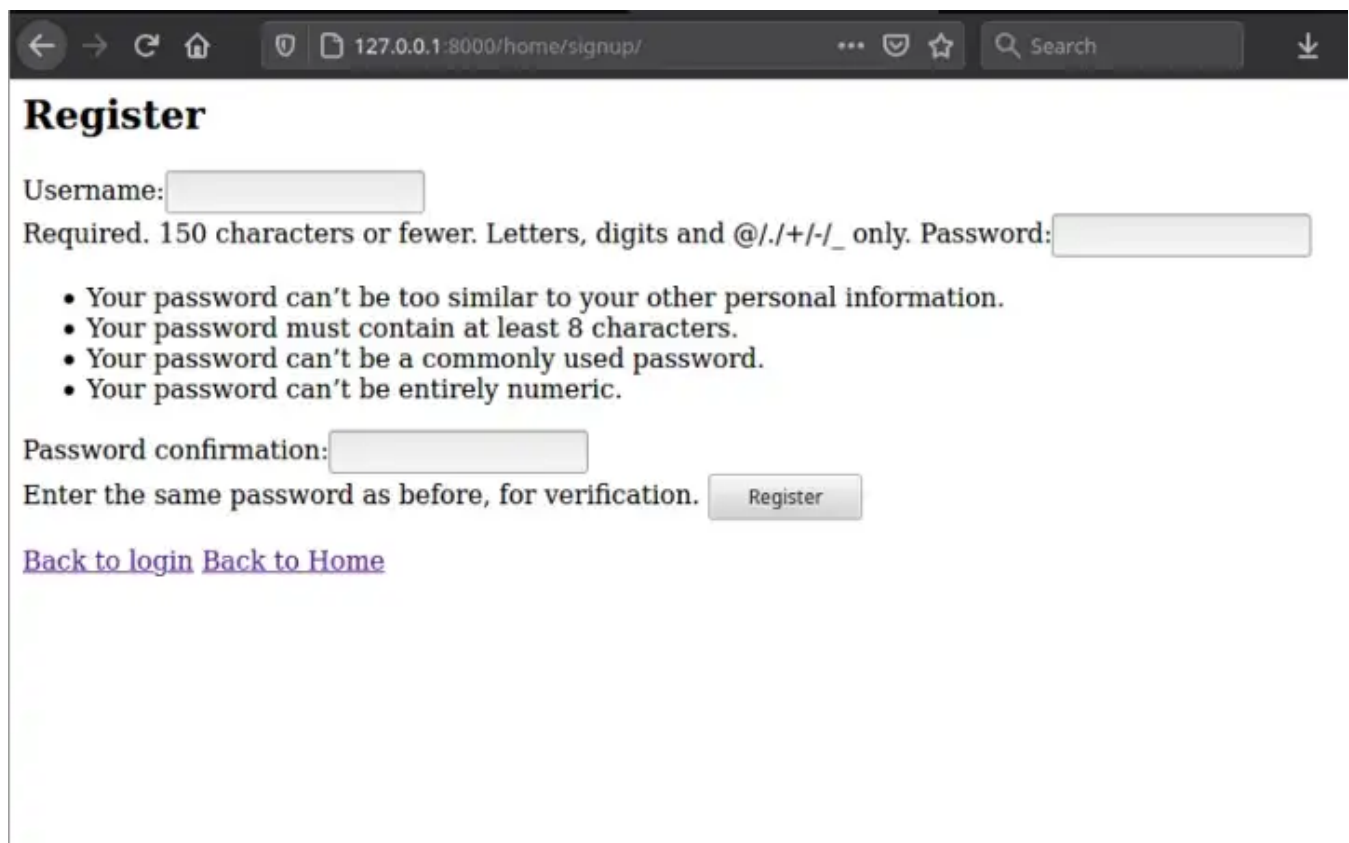
Lastly, add the view into `urls.py`. Remember to call `.as_view()` since it's a class-based view.

```
1 from django.urls import path
2 from . import views
3 urlpatterns = [
4
5     path('', views.home, name = "home"),
6     path("signup/", views.SignUp.as_view(), name="signup"),
7 ]
```

sign_url.py hosted with ❤ by GitHub

[view raw](#)

The registration page now looks like this:



Register

Username:

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only. Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation:

Enter the same password as before, for verification.

[Back to login](#) [Back to Home](#)

Registration page

After successful registration, a new user will be redirected to the login page.

Clean Up

Now let's do some clean-up to ensure that if a user is logged in, they see the logout button and Home link. Open `home.html` and add the following logic:

```
1  <html>
2    <head>
3      <style>
4        body {
5          color: pink;
6          text-align: center;
7        }
8
9      </style>
10   </head>
11   <body>
12
13     <h1>Hello, {{ user.username|default:'Guest' }}</h1>
14
15     <p> Welcome to the Our site</p>
16     <div>
17       {% if user.is_authenticated %}
18         <a href="{% url 'logout' %}">Logout</a>
19
20       {% else %}
21         <a href="{% url 'signup' %}">Sign Up</a>
22         <a href="{% url 'login' %}">Login</a>
23       {% endif %}
24     </div>
25
26   </body>
27 </html>
```

home.html hosted with ♥ by GitHub

[view raw](#)

Conclusion

Django provides a more effortless and built-in way to log in and authenticate users. You can also use the `UserCreationForm` provided by Django to create new users. I hope this tutorial will help you get started in designing your own login and registration system in Django. The full source code can be found [here](#).

More content at [PlainEnglish.io](https://python.plainenglish.io). Sign up for our [free weekly newsletter](#). Follow us on [Twitter](#), [LinkedIn](#), [YouTube](#), and [Discord](#).

[Programming](#)[Technology](#)[Software Engineering](#)[Data Science](#)[Machine Learning](#)

Enjoy the read? Reward the writer.^{Beta}

Your tip will go to Esther Vaati through a third-party platform of their choice, letting them know you appreciate their story.

[Give a tip](#)[About](#)[Help](#)[Terms](#)[Privacy](#)

Get the Medium app

