

You have 2 free stories left this month. [Sign up and get an extra one for free.](#)

How to Implement Login, Logout, and Registration with Django's User Model.



Kristian Roopnarine

Follow

Apr 7 · 8 min read ★

Using Django's built in User model to create a registration and login page.

The Django logo, featuring the word "django" in a white, lowercase, sans-serif font on a dark green background.

What you'll learn

- Django's built-in User model
- How to create Users with Django's built-in UserCreationForm
- Django's built-in authentication URL paths
- How to login and logout a user with Django's authentication URLs
- How to set permissions on Django views using decorators
- Documentation for Django's authentication system
- Working boilerplate for User authentication in Django

Django's User model

Django comes pre-built with features so that the programmer can focus on building a web app. An important aspect for any website is storing information about your users, to which Django provides us with a `User` model. The documentation can be found:

<https://docs.djangoproject.com/en/3.0/ref/contrib/auth/> and to get access to this model we can add this import:

```
from django.contrib.auth.models import User
```

Now we have access to all the amenities of the User model without needing to code much of it. The `User` model allows us to store information such as username, password, `first_name`, `last_name`, and even provides fields to check the permissions on that `User`.

If the default fields aren't enough for your application or you want to customize it yourself, Django allows you to do that as well by creating a model and inheriting from

`AbstractBaseUser`. The documentation on creating a custom user can be found here:

<https://docs.djangoproject.com/en/3.0/topics/auth/customizing/#reusable-apps-and-auth-user-model>.

User Registration with UserCreationForm

Django comes built with `ModelForms` which allow us to create forms directly from our models in a couple of lines of code and we can do the same for the `User` model called a `UserCreationForm`.

The form will include three fields by default, `username`, `password1`, and `password2`. The beauty of this form is in all the code that comes with it. The form will hash passwords to the database, handle verifying passwords, ensure that duplicate users will not be created and generate form errors. I know this seems too good to be true but it gets even better, these forms are highly customizable. You can add more fields, you can add custom logic to the forms `save()` method, you can add custom logic for form field verification and there's definitely much more because I've personally only begun to scratch the surface with Django model forms.

We can import the `UserCreationForm` and all it's goodness using:

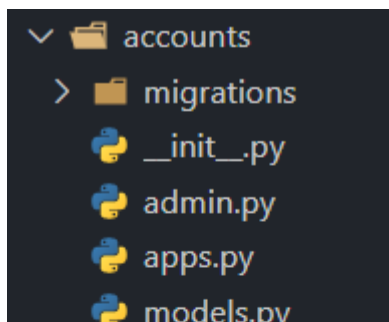
```
from django.contrib.auth.forms import UserCreationForm
```

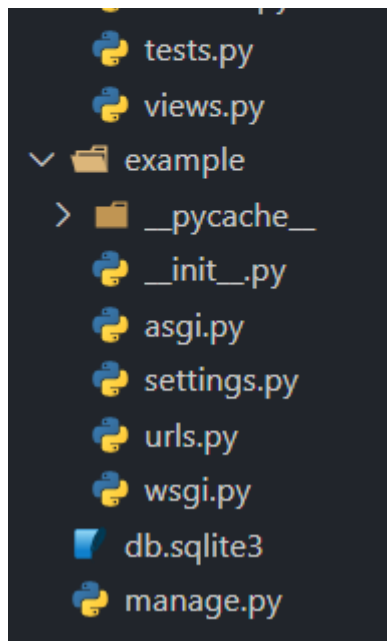
This module contains many other forms such as `PasswordChangeForm`, `UserChangeForm` and much more that can be found in the documentation, <https://docs.djangoproject.com/en/3.0/topics/auth/default/#creating-users>.

Let's get an example project running to show how powerful and easy to use this form is.

Django boilerplate

I created a Django project called `example` and created an app called `accounts`. I suggest recreating this structure for your Django projects because it will separate the logic and code related to the `User` model to one place.





Django boilerplate for accounts

Make sure to apply migrations to create the tables to store the `user` information.

Next we'll add `accounts` to our project `example` and create a `urls.py` file in the `accounts` app, and finally we need to include some URL paths into `example.urls`:

```
example/urls.py
```

```
from django.contrib import admin
```

```
from django.urls import path, include
```

```
urlpatterns = [
```

```
    path('admin/', admin.site.urls),
```

```
    path('', include("accounts.urls")),
```

```
    path('accounts/', include('django.contrib.auth.urls'))
```

```
]
```

The last URL path is very important! Django provides us with authentication views to handle many features including logging in and out a user. They don't provide us with a

template for these views and we'll add those later because Django searches for them in a very specific directory.

Now that we're done with configuring our project let's create some views.

In `accounts.views` add the following imports:

accounts/views.py

```
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth import login
from django.contrib.auth.decorators import login_required
```

The first two lines import our `User` model and model form `UserCreationForm`. The third line imports the `login` function which we'll use to authenticate our user after they've created an account and the fourth line imports a decorator that will redirect our user to the login form if they're not authenticated.

We'll create two views:

1. The view to render the `UserCreationForm`
2. A view displayed once the user is authenticated

The views for the login and logout will be handled by Django's authentication system.

Let's add the following code to the `accounts.views` file:

accounts/views.py

```
@login_required
def index(request):
    return render(request, 'accounts/index.html')

def sign_up(request):
    context = {}
    form = UserCreationForm(request.POST or None)
```

```

if request.method == "POST":
    if form.is_valid():
        user = form.save()
        login(request, user)
        return render(request, 'accounts/index.html')
context['form']=form
return render(request, 'registration/sign_up.html', context)

```

The `@login_required` decorator will check if our user is authenticated, and if not it will redirect them to “accounts/login” which is part of `django.contrib.auth.urls` that we included earlier. The `sign_up` view generates our form with empty information or the request.POST data if there is any. If our the `UserCreationForm` is being posted to this view, then we have to check if the fields are valid. The `form.is_valid()` verifies if the two passwords are the same by calling `validate_password()` which then calls `set_password()` to hash and save our password into the database.

Finally if everything is successful we log our user in with `login` and then redirects to the home page. Now that we’ve created our views we need to create the authentication templates and URL paths.

Authentication URL paths

Now that we’ve created some views we need to add these to our `urls.py` file in our `accounts` application. The `urls.py` file should look like:

```

accounts/urls.py

from django.contrib import admin

from django.urls import path

from . import views

urlpatterns = [

    path('', views.index, name="home"),

    path('accounts/sign_up/', views.sign_up, name="sign-up")

]

```

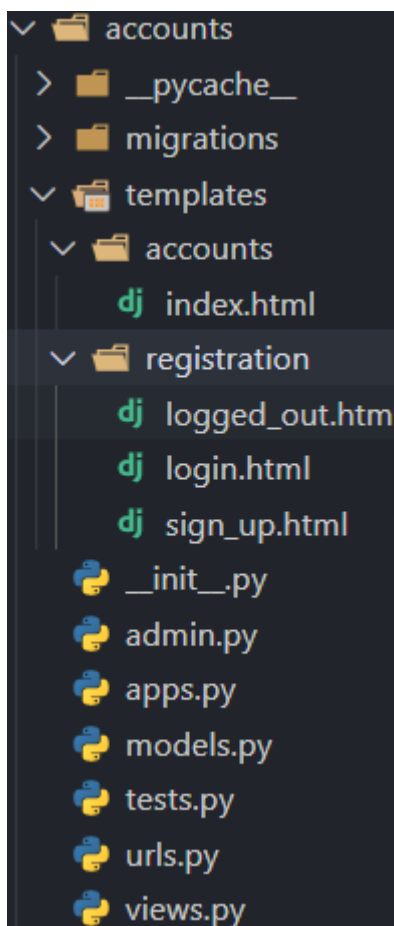
We've created our views and properly configured the URLs, which means the last thing to do is create our templates.

Authentication Templates

We need to create a `templates` directory inside of `accounts` that contains two sub-directories, `accounts` for rendering content after the user is logged in, and `registration` which handles all templates for authentication.

When using Django's authentication views it is very important to create a `registration` directory inside of your templates because it will search for the templates there! The documentation for authentication views can be found here, <https://docs.djangoproject.com/en/3.0/topics/auth/default/#module-django.contrib.auth.views>.

My `accounts` application directory now has the structure:



Accounts directory including templates

Within `templates` I created the sub-directories `accounts` and `registration`. The templates `logged_out.html` and `login.html` are two templates Django defaults to when using `django.contrib.auth.urls` for the URLs “accounts/logout” and “accounts/login” respectively. You can always customize these templates and how to do so can be found in the documentation.

Let's put the following code into our templates:

`index.html`:

```
index.html

<h1> Welcome {{user}} </h1>
```

`login.html`:

```
login.html

<form method="post">
    {% csrf_token %}

    {{form.as_p}}

    <input type="submit" value="Login">

</form>
```

`logged_out.html`:

```
logged_out.html

<h1> You are now logged out {{ user }} </h1>
```

`sign_up.html`:


```
sign_up.html

<form method="post">
    {% csrf_token %}

    {{form.as_p}}

    <input type="submit" value="Sign up">

</form>
```

The `{%csrf_token%}` is a security measure that helps prevents against Cross Site Request Forgery attacks and is required for every form in Django.

We don't need to include an action in our forms because Django automatically passes these parameters in depending on the form. If you need a specific redirect URL, you can configure this in the `settings.py` file which we'll include.

In the `settings.py` file include this at the bottom:

```
example/settings.py

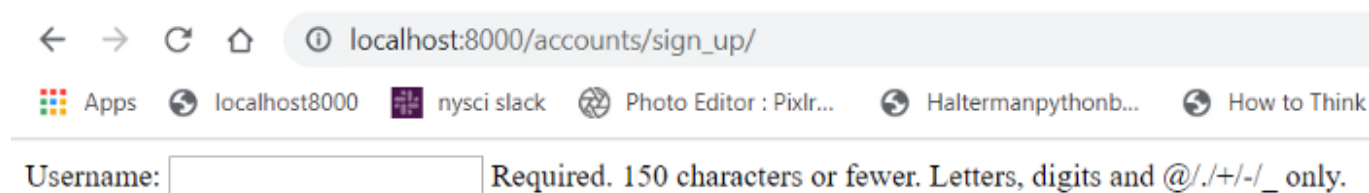
...

LOGIN_REDIRECT_URL = 'home'
```

This will tell Django to redirect to the URL name “home” after a successful login. Everything is ready so let's fire up our server and try everything out!

Our Django User authentication application

First let's visit our sign up page at `localhost:8000/accounts/sign_up` and you should see something similar to:



← → ↻ 🏠 ⓘ localhost:8000/accounts/sign_up/

📱 Apps 🌐 localhost8000 📄 nysci slack 🖼️ Photo Editor : Pixlr... 🐍 Haltermanpythonb... 📖 How to Think

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

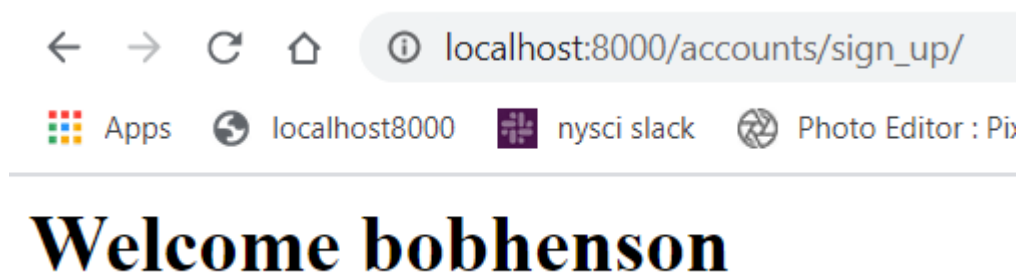
Password:

- 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.

UserCreationForm

We made a GET request to this URL which will render an empty `UserCreationForm`. It doesn't look that great but you can add custom styling to it, and all of this is being handled by our `UserCreationForm`. Create an account and you should be redirected to the home page.



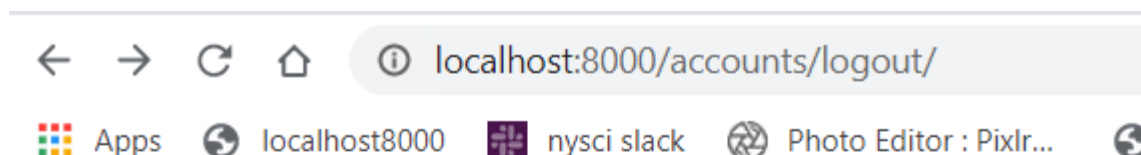
index.html after successful sign in

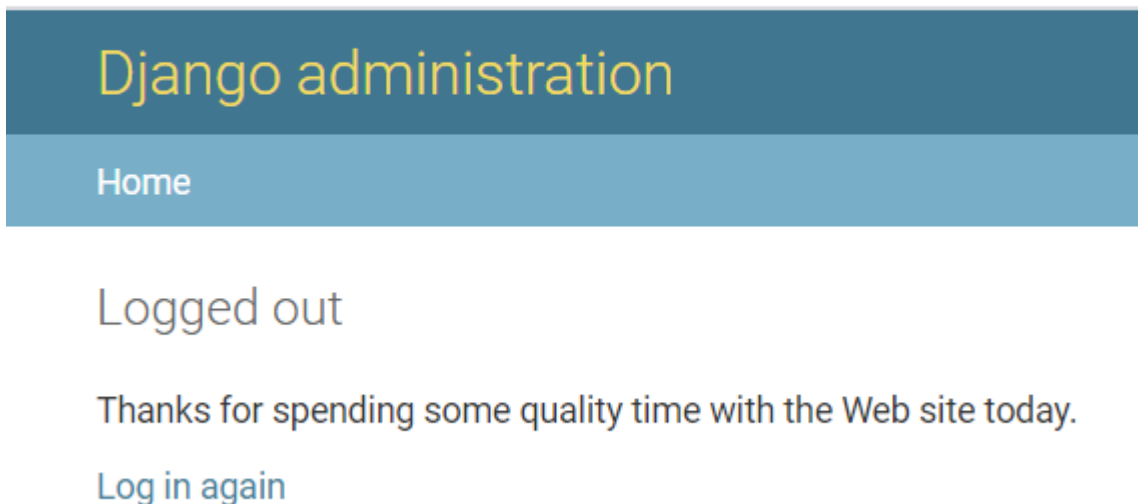
This successfully creates a User and stores it in the database. You can check it in the admin panel if you'd like. The `UserCreationForm` will handle any form input errors to prevent user duplication and validates the passwords.

Let's sign out of this application, and we can do that by simply going to the URL

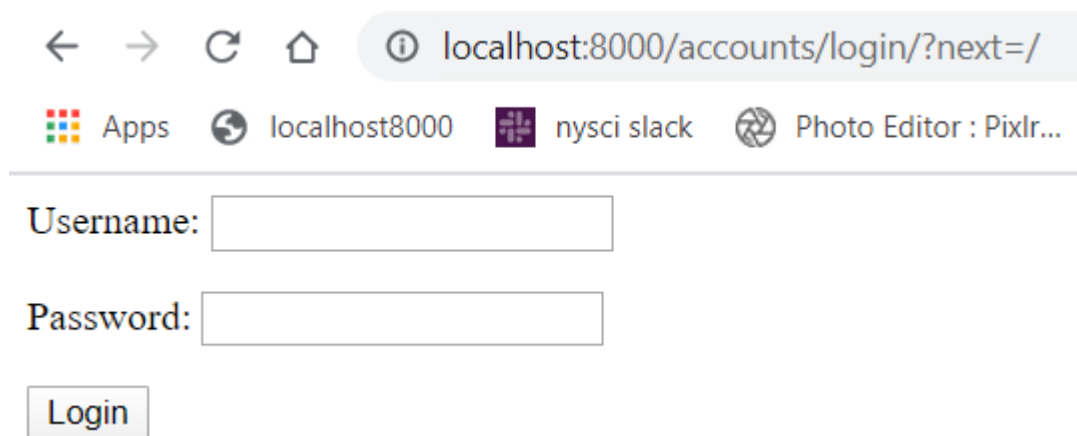
`localhost:8000/accounts/logout`.

Truthfully I didn't expect this outcome but it looks like our URL paths might have been conflicting with the admin site because when I visited the logout URL this is what I saw:





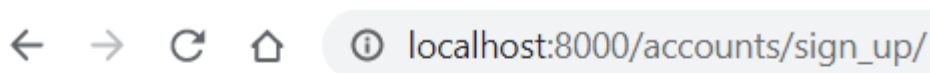
This isn't a huge issue because we can always change the logout URL directly. We can confirm we're logged out because if we try to visit the home page at `localhost:8000` we should be redirected to the login page.



Redirected from the Home page to Login page

Visiting the home page when we're not an authenticated user redirects us to the login page because of the `@login_required` decorator. You can see in the URL the `?next=` parameter that tells us where we'll be redirected to after a successful login, which we configured as the home page.

As expected, when I successfully logged in I was redirected to the home page.





Welcome bobhenson

Redirected after login

Wrap up

Now you have a working Django application with basic user authentication! You can create new users, log users in and out, and protect certain views from users that aren't logged in using decorators.

The boiler plate for this accounts application can be reused nearly anywhere, and I suggest learning how to customize the `UserCreationForm` and login form to tailor the look and feel for any application you create.

I'll create another post soon about authentication using Django REST Framework which will allow you to couple Django's authentication with front end frameworks like React.

[Django](#) [Python](#) [Web Development](#) [Programming](#)

[About](#) [Help](#) [Legal](#)

Get the Medium app

