Django Login

**Summary**: in this tutorial, you’ll learn how to create a Django login form that allows users to log in using a username and password.

# Create a new application

First, [create a new application](https://www.pythontutorial.net/django-tutorial/django-create-app/) called users by executing the startapp command:

django-admin startapp users

The project directory will look like this:

├── blog

├── db.sqlite3

├── manage.py

├── mysite

├── static

├── templates

└── users

Second, register the users application in the installed apps of the settings.py of the project:

INSTALLED\_APPS = [

'django.contrib.admin', 'django.contrib.auth', 'django.contrib.contenttypes', 'django.contrib.sessions',

'django.contrib.messages', 'django.contrib.staticfiles', *# local* 'blog.apps.BlogConfig',

'users.apps.UsersConfig'

]

Third, create a new urls.py file inside the users app with the following code:

from django.urls import path

from . import views

urlpatterns = []

Finally, include the urls.py of the users application in the urls.py of the project by using the include() function:

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls), path('',include('blog.urls')),

path('',include('users.urls'))

]

# Create a login form

First, create a login URL in the urls.py of the users application:

from django.urls import path

from . import views

urlpatterns = [

path('login/', views.sign\_in, name='login'),

]

Second, create a forms.py file in the users application and define the LoginForm that inherits from the Form class:

from django import forms

class LoginForm(forms.Form):

username = forms.CharField(max\_length=65)

password = forms.CharField(max\_length=65, widget=forms.PasswordInput)

The LoginForm has two fields username and password.

Third, create the sign\_in() function in the views.py file of the users application to render the login.html template:

from django.shortcuts import render

from .forms import LoginForm

def sign\_in(request):

if request.method == 'GET': form = LoginForm()

return render(request, 'users/login.html', {'form': form})

Fourth, create the templates/users directory inside the users application:

mkdir templates cd templates mkdir users

Fifth, create the login.html template inside the templates/users directory that extends the

base.html template:

{% extends 'base.html' %}

{% block content %}

<form method="POST" novalidate>

{% csrf\_token %}

<h2>Login</h2>

{{form.as\_p}}

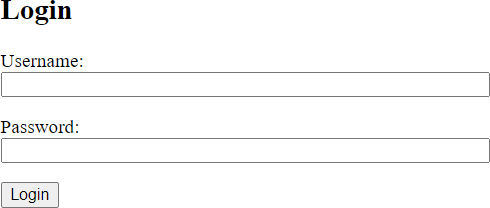
<input type="submit" value="Login" />

</form>

{% endblock content%}

Sixth, open the login URL, and you’ll see the login form:

http://127.0.0.1:8000/login



If you enter a username/password and click the Login button, you’ll get an error because we haven’t added the code that handles the HTTP POST request yet.

Seventh, modify the sign\_in() function to handle the login process:

from django.shortcuts import render, redirect

from django.contrib import messages

from django.contrib.auth import login, authenticate from .forms import LoginForm

def sign\_in(request):

if request.method == 'GET':

form = LoginForm()

return render(request,'users/login.html', {'form': form})

elif request.method == 'POST': form = LoginForm(request.POST)

if form.is\_valid():

username = form.cleaned\_data['username'] password = form.cleaned\_data['password']

user = authenticate(request,username=username,password=password) if user:

login(request, user)

messages.success(request,f'Hi {username.title()}, welcome back!') return redirect('posts')

*# form is not valid or user is not authenticated* messages.error(request,f'Invalid username or password') return render(request,'users/login.html',{'form': form})

How it works.

First, import the authenticate and login function from the django.contrib.auth module:

from django.contrib.auth import login, authenticate

The authenticate() function verifies a username and password. If the username and password are valid, it returns an instance of User class or None otherwise.

The login() function logs a user in. Technically, it creates a session id on the server and sends it back to the web browser in the form of a cookie.

In the subsequent request, the web browser sends the session id back to the web server, Django matches the cookie value with the session id and creates the User object.

Second, verify the username and password using the authenticate() function if the form is valid:

user = authenticate(request, username=username, password=password)

Third, log the user in, [create a flash message](https://www.pythontutorial.net/django-tutorial/django-flash-messages/), and redirect the user to the posts URL if the username and password are valid:

if user:

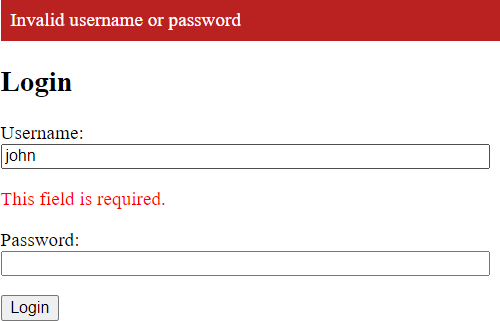
login(request, user)

messages.success(request,f'Hi {user.username.title()}, welcome back!') return redirect('posts')

Otherwise, create a flash error message and redirect the user back to the login page:

messages.error(request,f'Invalid username or password') return render(request,'users/login.html')

If you enter a username without a password and click the Login button, you’ll get the following error message:



However, if you enter the correct username/password, you’ll be redirected to the post list page with a welcome message:



# Add a Logout form

First, define a route for logging a user out:

from django.urls import path

from . import views

urlpatterns = [

path('login/', views.sign\_in, name='login'), path('logout/', views.sign\_out, name='logout'),

]

Second, define the sign\_out() function in the views.py file that handles the logout route:

from django.shortcuts import render, redirect from django.contrib import messages

from django.contrib.auth import login, authenticate, logout from .forms import LoginForm

def sign\_in(request):

if request.method == 'GET': form = LoginForm()

return render(request,'users/login.html', {'form': form})

elif request.method == 'POST': form = LoginForm(request.POST)

if form.is\_valid():

username = form.cleaned\_data['username'] password=form.cleaned\_data['password']

user = authenticate(request,username=username,password=password)

if user:

login(request, user)

messages.success(request,f'Hi {username.title()}, welcome back!') return redirect('posts')

*# either form not valid or user is not authenticated* messages.error(request,f'Invalid username or password') return render(request,'users/login.html',{'form': form})

def sign\_out(request): logout(request)

messages.success(request,f'You have been logged out.') return redirect('login')

If you log in and access the login page, you’ll still see the login form. Therefore, it’s better to redirect the logged user to the post list instead if the user accesses the login page.

Third, modify the sign\_in() function in the views.py of the users application:

def sign\_in(request):

if request.method == 'GET':

if request.user.is\_authenticated: return redirect('posts')

form = LoginForm()

return render(request,'users/login.html', {'form': form})

elif request.method == 'POST': form = LoginForm(request.POST)

if form.is\_valid():

username = form.cleaned\_data['username'] password=form.cleaned\_data['password']

user = authenticate(request,username=username,password=password)

if user:

login(request, user)

messages.success(request,f'Hi {username.title()}, welcome back!') return redirect('posts')

*# either form not valid or user is not authenticated* messages.error(request,f'Invalid username or password') return render(request,'users/login.html',{'form': form})

The the request.user.is\_authenticated returns True if a user is logged in or False

otherwise.

Fourth, modify the base.html template to include the logout link if the user is authenticated and the login link otherwise:

{%load static %}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<link rel="stylesheet" href="{% static 'css/style.css' %}" />

<script src="{% static 'js/app.js' %}" defer></script>

<title>My Site</title>

</head>

<body>

<header>

{%if request.user.is\_authenticated %}

<span>Hi {{ request.user.username | title }}</span>

<a href="{% url 'logout' %}">Logout</a>

{%else%}

<a href="{% url 'login' %}">Login</a>

{%endif%}

</header>

<main>

{% if messages %}

<div class="messages">

{% for message in messages %}

<div class="alert {% if message.tags %}alert-{{ messa

{{ message }}

</div>

{% endfor %}

</div>

{% endif %}

{%block content%}

{%endblock content%}

</main>

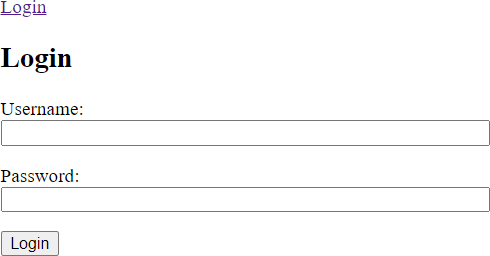
</body>

</html>

If you access the site, you’ll see the login link:



When you click the login link, it’ll open the login page:



If you enter the valid username and password and log in, you’ll see a welcome message as well as the logout link:



If you click the logout link, it redirects to the login page:



# Hiding the edit and delete links on the post list

If a user is logged in, the request.user.is\_authenticated is True . Therefore, you can use this object to show and hide elements of the page whether the user is logged in or not.

For example, the following hides the editing and deleting links on the blog/home.html

template if the user is authenticated:



{% extends 'base.html' %}

{% block content %}

<h1>My Posts</h1>

{% for post in posts %}

<h2>{{ post.title }}</h2>

<small>Published on {{ post.published\_at | date:"M d, Y" }} by {{ pos

<p>{{ post.content }}</p>

{% if request.user.is\_authenticated %}

<p>

<a href="{% url 'post-edit' post.id %}">Edit</a>

<a href="{% url 'post-delete' post.id%}">Delete</a>

</p>

{% endif %}

{% endfor %}

{% endblock content %}

# Protecting the protected pages

Typically, you should allow authenticated users to access the creating, updating, and deleting post pages. To do that you can use Django’s login\_required decorator.

If a view function has the login\_required decorator and an unauthenticated user attempts to run it, Django will redirect the user to the login page.

We’ll protect the create, update, and delete post functions using the login\_required

decorator.

First, set the login URL in the settings.py of to the login URL:

LOGIN\_URL = 'login'

If you don’t do this, Django will redirect to the default login URL which is accounts/login/ not

users/login as we defined in this project.

Second, modify the views.py of the blog application by adding the @login\_required

decorator to the create\_post , edit\_post , and delete\_post functions:

from django.shortcuts import render, redirect, get\_object\_or\_404 from django.contrib import messages

from django.contrib.auth.decorators import login\_required

from .models import Post from .forms import PostForm

@login\_required

def delete\_post(request, id):

post = get\_object\_or\_404(Post, pk=id) context = {'post': post}

if request.method == 'GET':

return render(request, 'blog/post\_confirm\_delete.html', context) elif request.method == 'POST':

post.delete()

messages.success(request, 'The post has been deleted successfully.') return redirect('posts')

@login\_required

def edit\_post(request, id):

post = get\_object\_or\_404(Post, id=id)

if request.method == 'GET':

context = {'form': PostForm(instance=post), 'id': id} return render(request, 'blog/post\_form.html', context)

elif request.method == 'POST':

form = PostForm(request.POST, instance=post) if form.is\_valid():

form.save() messages.success(

request, 'The post has been updated successfully.') return redirect('posts')

else:

messages.error(request, 'Please correct the following errors:') return render(request, 'blog/post\_form.html', {'form': form})

@login\_required

def create\_post(request):

if request.method == 'GET': context = {'form': PostForm()}

return render(request, 'blog/post\_form.html', context)

elif request.method == 'POST': form = PostForm(request.POST) if form.is\_valid():

form.save()

messages.success(

request, 'The post has been created successfully.') return redirect('posts')

else:

messages.error(request, 'Please correct the following errors:') return render(request, 'blog/post\_form.html', {'form': form})

def home(request):

posts = Post.objects.all() context = {'posts': posts}

return render(request, 'blog/home.html', context)

def about(request):

return render(request, 'blog/about.html')

If you open the create, update, or delete URL, for example:

[http:*//127.0.0.1/post/create*](http://127.0.0.1/post/create)

It’ll be redirected to the login page.

# Summary

Use authenticate() function to verify a user by username and password. Use login() function to log a user in.

Use logout() function to log a user out.

Use request.user.is\_authenticated to check if the current user is authenticated. User @login\_required decorator to protect pages from unauthenticated users.