Django Registration

**Summary**: in this tutorial, you’ll learn how to create a Django registration form that allows users to sign up.

# Creating a Django registration form

First, define a registration URL in the urls.py of the users app:

from django.urls import path

from . import views

urlpatterns = [

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

]

Second, define a RegisterForm class in the forms.py of the users.py file:

from django import forms

from django.contrib.auth.models import User

from django.contrib.auth.forms import UserCreationForm

class LoginForm(forms.Form):

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

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

class RegisterForm(UserCreationForm): class Meta:

model=User

fields = ['username','email','password1','password2']

The RegisterForm uses the built-in User model and includes four fields including username ,

email , password1 , and password2 , which the user needs to fill in for registration. Third, define the sign\_up() view function in the views.py of the users app:

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

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

from .forms import LoginForm, RegisterForm

def sign\_up(request):

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

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

The sign\_up() view function creates the RegisterForm object and renders it in the

register.html template.

Fourth, create a register.html template in the templates/users directory of the users

application:

{% extends 'base.html' %}

{% block content %}

<form method="POST" novalidate>

{% csrf\_token %}

<h2>Sign Up</h2>

{{ form.as\_p }}

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

</form>

{% endblock content%}

The registerl.html extends the base.html template of the project. It renders the

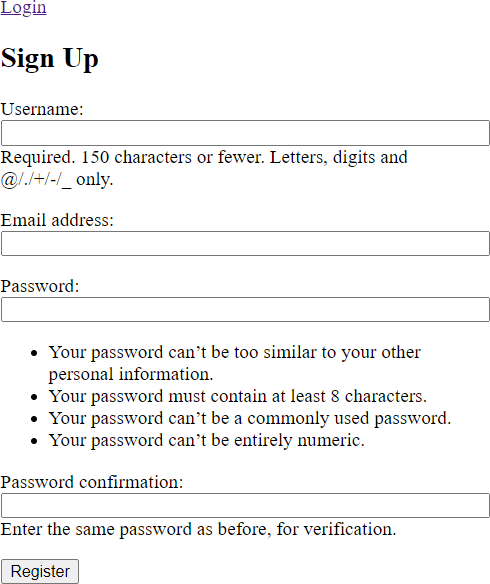
RegisterForm (form).

Note that the novalidate property removes the HTML5 validation. Once completing the project, you can remove this property to enable HTML5 validation.

Finally, open the registration URL:

http://127.0.0.1:8000/register/

…you’ll see the registration form as follows:



# Customize the Django register form

The registration form has four fields with a lot of information. This information comes from the default User model.

If you want to customize the information displayed on the form, you can modify the

register.html template as follows:

{% extends 'base.html' %}

{% block content %}

<form method="POST" novalidate>

{% csrf\_token %}

<h2>Sign Up</h2>

{% for field in form %}

<p>

{% if field.errors %}

<ul class="errorlist">

{% for error in field.errors %}

<li>{{ error }}</li>

{% endfor %}

</ul>

{% endif %}

{{ field.label\_tag }} {{ field }}

</p>

{% endfor %}

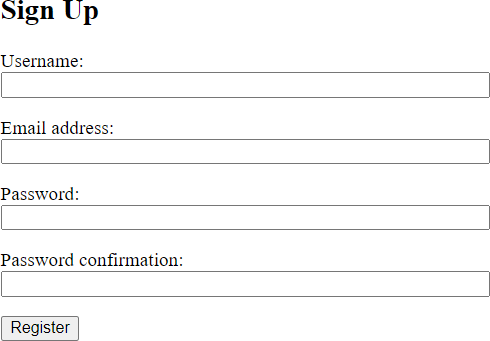
<input type="submit" value="Register" />

</form>

{% endblock content%}

The template iterates over fields of the form and outputs each field individually. For each field, it displays the error list if the validation fails.

The new form will look like this:



If you fill out the information and click register, you’ll get an error because we haven’t added the code that handles the HTTP POST request.

# Handling Registration logic

To handle the HTTP POST request, you modify the sign\_up() function in the views.py file of the users application:

def sign\_up(request):

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

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

if request.method == 'POST':

form = RegisterForm(request.POST) if form.is\_valid():

user = form.save(commit=False)

user.username = user.username.lower() user.save()

messages.success(request, 'You have singed up successfully.') login(request, user)

return redirect('posts')

else:

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

How it works.

First, create a new instance of the RegisterForm :

form = RegisterForm(request.POST)

If the form is valid, we save the form but do not immediately store it in the database. This is done by passing the argument commit=False to the save() method of the form object.

The reason is that we want to make the user name lowercase before saving it in the database:

user.username = user.username.lower() user.save()

After saving the user, we [create a flash message](https://www.pythontutorial.net/django-tutorial/django-flash-messages/), [log the user in](https://www.pythontutorial.net/django-tutorial/django-login/) and redirect the user to the post list page:

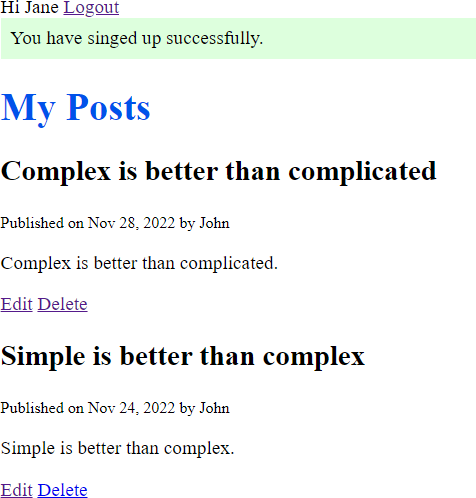
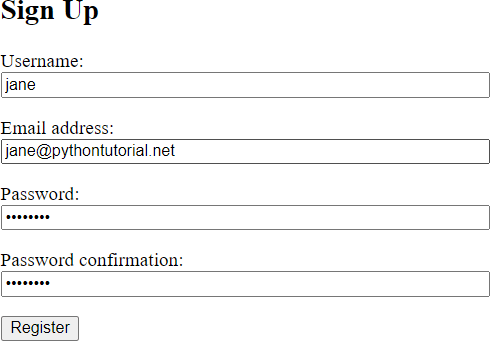
messages.success(request, 'You have singed up successfully.') login(request, user)

return redirect('posts')

If the form is not valid, we rerender the form with the previously entered values by passing the form object to the render() function:

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

The following example illustrates how to sign up a user with the username jane :



# Adding the links

First, include the registration link by modifying the base.html template. Also, include the My Posts and New Post links if the user is logged in:

{%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 %}

<a href="{% url 'posts' %}">My Posts</a>

<a href="{% url 'post-create' %}">New Post</a>

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

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

{%else%}

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

<a href="{% url 'register' %}">Register</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>

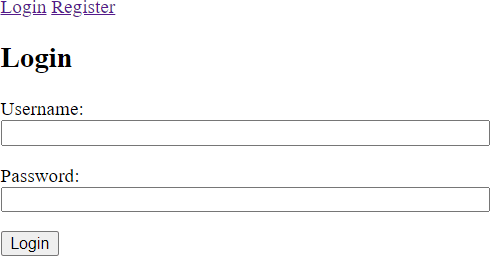
{% endif %}

{%block content%}

{%endblock content%}

</main>

</body>



Second, include the registration link in the login.html template:

{% extends 'base.html' %}

{% block content %}

<form method="POST" novalidate>

{% csrf\_token %}

<h2>Login</h2>

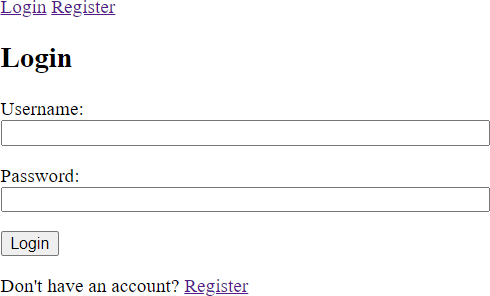
{{form.as\_p}}

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

<p>Don't have an account? <a href="{%url 'register' %}">Register</a></p>

</form>

{% endblock content%}



Third, add the login link to the register.html page:

{% extends 'base.html' %}

{% block content %}

<form method="POST" novalidate>

{% csrf\_token %}

<h2>Sign Up</h2>

{% for field in form %}

<p>

{% if field.errors %}

<ul class="errorlist">

{% for error in field.errors %}

<li>{{ error }}</li>

{% endfor %}

</ul>

{% endif %}

{{ field.label\_tag }} {{ field }}

</p>

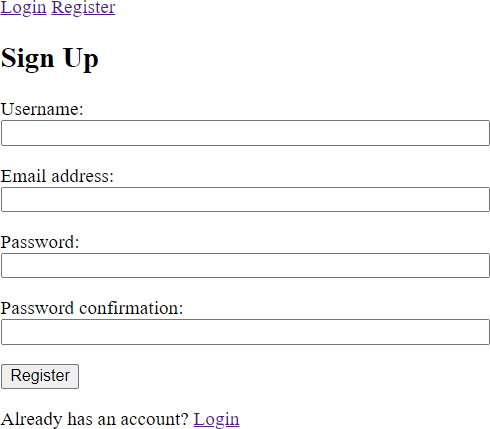
{% endfor %}

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

<p>Already has an account? <a href="{%url 'login' %}">Login</a></p>

</form>

{% endblock content%}



# Preventing a user from editing / deleting posts of other users

Typically, a user should not be able to edit or delete a post of other users. However, the user can view the posts from other users.

To implement this function, we need to check if the author of the post is the same as the

currently logged user. If yes, we render the edit/delete forms. Otherwise, we can redirect the users to a 404 page.

Also, we need to hide the edit and delete links of the post list page if the posts do not belong to the user.

First, modify the views.py of the blog application:

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):

queryset = Post.objects.filter(author=request.user) post = get\_object\_or\_404(queryset, 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):

queryset = Post.objects.filter(author=request.user) post = get\_object\_or\_404(queryset, pk=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)

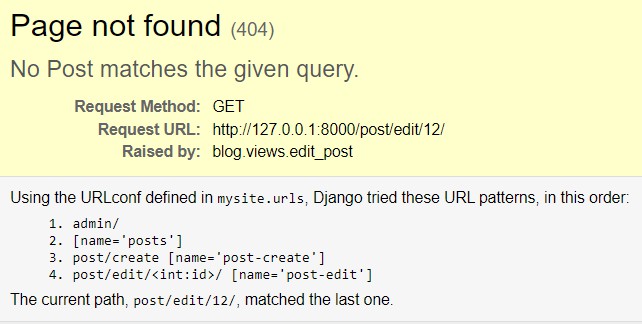
In both delete and update, we filter the post by the current user before passing it to the

get\_object\_or\_404() function.

If you log in as Jane and attempt to edit a post that does not belong to Jane, you’ll get a 404 error. For example:



The 404 page:



Second, modify the home.html template to hide the edit and delete links.



{% 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 and request.user == post.author %

<p>

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

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

</p>

{% endif %}

{% endfor %}

{% endblock content %}

# Removing the author from the create post form

First, remove the author field from the fields of the PostForm class:

from django.forms import ModelForm

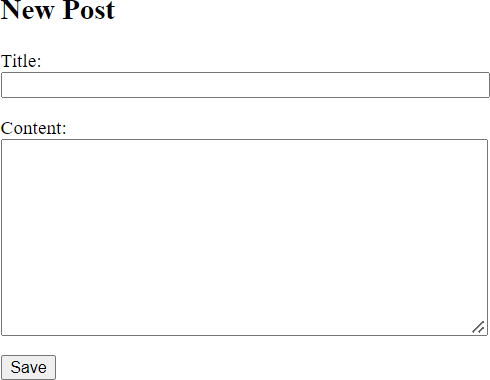
from .models import Post

class PostForm(ModelForm): class Meta:

model = Post

fields = ['title','content']

The post-creation form will look like this:



Second, modify the create\_post() function in the views.py of the blog application to update the author of the post to the currently logged user:

@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():

user = form.save(commit=False)

user.author = request.user user.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})

# Summary

 Subclass the UserCreationForm to create a custom Django registration form.