# Django 搭建个人博客系统

项目名称:MyBlog 应用名称:Blog(python manage.py startapp Blog)

**#MyBlog/settings**

import os  
  
from pathlib import Path  
  
# Build paths inside the project like this: BASE\_DIR / 'subdir'.  
BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent  
  
  
# Quick-start development settings - unsuitable for production  
# See https://docs.djangoproject.com/en/5.1/howto/deployment/checklist/  
  
# SECURITY WARNING: keep the secret key used in production secret!  
SECRET\_KEY = 'django-insecure-$5\_04!-452+e3c7y(09\*y\_nt(+g3=ss$3o\*x1wwlk4$5ayc!uk'  
  
# SECURITY WARNING: don't run with debug turned on in production!  
DEBUG = True  
  
ALLOWED\_HOSTS = []  
  
  
# Application definition  
  
INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
 'Blog',  
]  
  
MIDDLEWARE = [  
 'django.middleware.security.SecurityMiddleware',  
 'django.contrib.sessions.middleware.SessionMiddleware',  
 'django.middleware.common.CommonMiddleware',  
 'django.middleware.csrf.CsrfViewMiddleware',  
 'django.contrib.auth.middleware.AuthenticationMiddleware',  
 'django.contrib.messages.middleware.MessageMiddleware',  
 'django.middleware.clickjacking.XFrameOptionsMiddleware',  
]  
  
ROOT\_URLCONF = 'MyBlog.urls'  
  
TEMPLATES = [  
 {  
 'BACKEND': 'django.template.backends.django.DjangoTemplates',  
 'DIRS': [BASE\_DIR / 'templates']  
 ,  
 'APP\_DIRS': True,  
 'OPTIONS': {  
 'context\_processors': [  
 'django.template.context\_processors.debug',  
 'django.template.context\_processors.request',  
 'django.contrib.auth.context\_processors.auth',  
 'django.contrib.messages.context\_processors.messages',  
 ],  
 },  
 },  
]  
  
WSGI\_APPLICATION = 'MyBlog.wsgi.application'  
  
  
# Database  
# https://docs.djangoproject.com/en/5.1/ref/settings/#databases  
  
DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.sqlite3',  
 'NAME': BASE\_DIR / 'db.sqlite3',  
 }  
}  
  
  
# Password validation  
# https://docs.djangoproject.com/en/5.1/ref/settings/#auth-password-validators  
  
AUTH\_PASSWORD\_VALIDATORS = [  
 {  
 'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',  
 },  
]  
  
  
# Internationalization  
# https://docs.djangoproject.com/en/5.1/topics/i18n/  
  
LANGUAGE\_CODE = 'en-us'  
  
TIME\_ZONE = 'UTC'  
  
USE\_I18N = True  
  
USE\_TZ = True  
  
  
# Static files (CSS, JavaScript, Images)  
# https://docs.djangoproject.com/en/5.1/howto/static-files/  
  
STATIC\_URL = 'static/'  
  
# Default primary key field type  
# https://docs.djangoproject.com/en/5.1/ref/settings/#default-auto-field  
  
DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'  
  
# 登录成功后重定向到文章列表页  
LOGIN\_REDIRECT\_URL = '/blog/'  
  
# 登录页面的 URL  
LOGIN\_URL = '/login/'  
  
# MyBlog/settings.py  
  
# 登出后重定向到首页  
LOGOUT\_REDIRECT\_URL = '/blog/' # 确保这个 URL 是有效的

**#models**

from django.db import models  
from django.contrib.auth.models import User  
  
# 自定义模型管理器  
class PostManager(models.Manager):  
 def get\_posts\_by\_user(self, user):  
 *"""获取某个用户的所有文章"""* return self.filter(author=user)  
  
class Post(models.Model):  
 title = models.CharField(max\_length=200, null=False)  
 content = models.TextField()  
 author = models.ForeignKey(User, on\_delete=models.CASCADE)  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
 updated\_at = models.DateTimeField(auto\_now=True)  
  
 objects = PostManager() # 使用自定义管理器  
  
 def \_\_str\_\_(self):  
 return self.title  
  
 class Meta:  
 ordering = ['-created\_at']

**#主urls(MyBlog/urls)**

# MyBlog/urls.py  
from django.contrib import admin  
from django.urls import path, include  
from django.views.generic import RedirectView  
from django.contrib.auth import views as auth\_views # 导入 Django 内置的认证视图  
  
# 定义项目的 URL 路由规则  
urlpatterns = [  
 path('admin/', admin.site.urls), # 管理员界面的路由  
 path('blog/', include('Blog.urls')), # 包含 Blog 应用的 URL 路由  
 path('', RedirectView.as\_view(url='/blog/')), # 将根路径重定向到 /blog/  
  
 # 添加 Django 内置的登录和注销视图  
 path('login/', auth\_views.LoginView.as\_view(template\_name='registration/login.html'), name='login'),  
 path('logout/', auth\_views.LogoutView.as\_view(), name='logout'),  
]

**#次urls(Blog/urls)**

from django.urls import path  
from . import views  
  
# 定义URL模式，将URL映射到视图函数  
urlpatterns = [  
 # 主页，显示帖子列表  
 path('', views.post\_list, name='post\_list'),  
 # 帖子详情页，通过帖子ID获取详细信息  
 path('<int:post\_id>/', views.post\_detail, name='post\_detail'),  
 # 创建新帖子页面  
 path('create/', views.post\_create, name='post\_create'),  
 # 更新帖子页面，通过帖子ID确定要更新的帖子  
 path('<int:post\_id>/update/', views.post\_update, name='post\_update'),  
 # 删除帖子页面，通过帖子ID确定要删除的帖子  
 path('<int:post\_id>/delete/', views.post\_delete, name='post\_delete'),  
 # 用户帖子列表页面，显示当前用户的所有帖子  
 path('my-posts/', views.user\_posts, name='user\_posts'), # 新增路由  
]

**#Blog/views**

from django.shortcuts import render, get\_object\_or\_404, redirect  
from django.core.paginator import Paginator  
from django.contrib import messages  
from django.contrib.auth.decorators import login\_required  
from .models import Post  
from .forms import PostForm  
from django.http import JsonResponse  
from django.views.decorators.csrf import csrf\_exempt  
from django.core.files.storage import default\_storage  
  
# 文章列表  
def post\_list(request):  
 *"""  
 显示文章列表页面，分页显示所有文章。  
 参数:  
 - request: HttpRequest对象，包含用户请求的信息。  
 返回:  
 - HttpResponse对象，包含渲染后的文章列表页面。  
 """* # 获取所有文章，按创建时间倒序排列  
 posts = Post.objects.all().order\_by('-created\_at')  
 # 初始化分页器，每页显示5篇文章  
 paginator = Paginator(posts, 5)  
 # 获取当前页码  
 page\_number = request.GET.get('page')  
 # 获取当前页码的文章对象  
 page\_obj = paginator.get\_page(page\_number)  
 # 渲染文章列表页面  
 return render(request, 'post\_list.html', {'page\_obj': page\_obj})  
  
# 文章详情  
def post\_detail(request, post\_id):  
 *"""  
 显示文章详情页面。  
 参数:  
 - request: HttpRequest对象，包含用户请求的信息。  
 - post\_id: 文章ID，用于获取指定文章。  
 返回:  
 - HttpResponse对象，包含渲染后的文章详情页面。  
 """* # 获取指定文章对象，如果不存在则返回404错误  
 post = get\_object\_or\_404(Post, id=post\_id)  
 # 渲染文章详情页面  
 return render(request, 'post\_detail.html', {'post': post})  
  
# 创建文章  
@login\_required  
def post\_create(request):  
 *"""  
 创建新文章。  
 参数:  
 - request: HttpRequest对象，包含用户请求的信息。  
 返回:  
 - HttpResponse对象，包含渲染后的文章创建表单页面或重定向到文章详情页面。  
 """* if request.method == 'POST':  
 # 创建PostForm实例，用POST数据填充表单  
 form = PostForm(request.POST)  
 if form.is\_valid():  
 # 保存表单但不提交到数据库  
 post = form.save(commit=False)  
 # 设置文章作者为当前用户  
 post.author = request.user  
 # 保存文章到数据库  
 post.save()  
 # 显示成功消息  
 messages.success(request, '文章已成功创建！')  
 # 重定向到新创建的文章详情页面  
 return redirect('post\_detail', post\_id=post.id)  
 else:  
 # 如果不是POST请求，则创建空表单  
 form = PostForm()  
 # 渲染文章创建表单页面  
 return render(request, 'post\_create.html', {'form': form})  
  
# 编辑文章  
@login\_required  
def post\_update(request, post\_id):  
 *"""  
 编辑指定文章。  
 参数:  
 - request: HttpRequest对象，包含用户请求的信息。  
 - post\_id: 文章ID，用于获取指定文章。  
 返回:  
 - HttpResponse对象，包含渲染后的文章编辑表单页面或重定向到文章详情页面。  
 """* # 获取指定文章对象，如果不存在则返回404错误  
 post = get\_object\_or\_404(Post, id=post\_id)  
 # 检查当前用户是否有权限编辑文章  
 if post.author != request.user:  
 # 如果没有权限，返回403错误页面  
 return render(request, '403.html', status=403)  
 if request.method == 'POST':  
 # 创建PostForm实例，用POST数据填充表单并指定文章实例  
 form = PostForm(request.POST, instance=post)  
 if form.is\_valid():  
 # 保存编辑后的文章  
 form.save()  
 # 显示成功消息  
 messages.success(request, '文章已成功更新！')  
 # 重定向到编辑后的文章详情页面  
 return redirect('post\_detail', post\_id=post.id)  
 else:  
 # 如果不是POST请求，创建表单并填充文章数据  
 form = PostForm(instance=post)  
 # 渲染文章编辑表单页面  
 return render(request, 'post\_update.html', {'form': form})  
  
# 删除文章  
@login\_required  
def post\_delete(request, post\_id):  
 *"""  
 删除指定文章。  
 参数:  
 - request: HttpRequest对象，包含用户请求的信息。  
 - post\_id: 文章ID，用于获取指定文章。  
 返回:  
 - HttpResponse对象，包含渲染后的文章删除确认页面或重定向到文章列表页面。  
 """* # 获取指定文章对象，如果不存在则返回404错误  
 post = get\_object\_or\_404(Post, id=post\_id)  
 # 检查当前用户是否有权限删除文章  
 if post.author != request.user:  
 # 如果没有权限，返回403错误页面  
 return render(request, '403.html', status=403)  
 if request.method == 'POST':  
 # 删除文章  
 post.delete()  
 # 显示成功消息  
 messages.success(request, '文章已成功删除！')  
 # 重定向到文章列表页面  
 return redirect('post\_list')  
 # 渲染文章删除确认页面  
 return render(request, 'post\_delete.html', {'post': post})  
  
# 获取当前用户的所有文章  
@login\_required  
def user\_posts(request):  
 *"""  
 显示当前用户的所有文章。  
 参数:  
 - request: HttpRequest对象，包含用户请求的信息。  
 返回:  
 - HttpResponse对象，包含渲染后的用户文章列表页面。  
 """* # 使用自定义模型管理器获取当前用户的所有文章  
 user\_posts = Post.objects.get\_posts\_by\_user(request.user)  
 # 渲染用户文章列表页面  
 return render(request, 'user\_posts.html', {'user\_posts': user\_posts})  
  
# 上传图片  
@csrf\_exempt  
def upload\_image(request):  
 *"""  
 处理图片上传请求。  
 参数:  
 - request: HttpRequest对象，包含用户请求的信息。  
  
 返回:  
 - JsonResponse对象，包含上传图片的位置或错误信息。  
 """* if request.method == 'POST' and request.FILES.get('file'):  
 # 获取上传的图片文件  
 upload = request.FILES['file']  
 # 保存上传的图片文件  
 file\_name = default\_storage.save(f'uploads/{upload.name}', upload)  
 # 获取上传图片的URL  
 file\_url = default\_storage.url(file\_name)  
 # 返回上传图片的位置  
 return JsonResponse({'location': file\_url})  
 # 如果上传失败，返回错误信息  
 return JsonResponse({'error': '上传失败'}, status=400)  
  
  
# Blog/views.py  
def post\_list(request):  
 *"""  
 显示文章列表页面，分页显示所有文章。  
 参数:  
 - request: HttpRequest对象，包含用户请求的信息。  
 返回:  
 - HttpResponse对象，包含渲染后的文章列表页面。  
 """* print("post\_list view is called!") # 调试信息  
 # 获取所有文章，按创建时间倒序排列  
 posts = Post.objects.all().order\_by('-created\_at')  
 # 初始化分页器，每页显示5篇文章  
 paginator = Paginator(posts, 5)  
 # 获取当前页码  
 page\_number = request.GET.get('page')  
 # 获取当前页码的文章对象  
 page\_obj = paginator.get\_page(page\_number)  
 # 渲染文章列表页面  
 return render(request, 'post\_list.html', {'page\_obj': page\_obj})

**#新建Blog/forms**

from django import forms  
from .models import Post  
  
class PostForm(forms.ModelForm):  
 class Meta:  
 model = Post  
 fields = ['title', 'content']  
 widgets = {  
 'title': forms.TextInput(attrs={  
 'class': 'input',  
 'placeholder': '请输入文章标题',  
 'required': True # HTML5 前端验证  
 }),  
 'content': forms.Textarea(attrs={  
 'class': 'textarea',  
 'placeholder': '请输入文章内容',  
 'required': True # HTML5 前端验证  
 }),  
 }  
  
 def clean\_title(self):  
 title = self.cleaned\_data.get('title')  
 if not title:  
 raise forms.ValidationError("标题不能为空！")  
 if len(title) > 200:  
 raise forms.ValidationError("标题长度不能超过 200 个字符！")  
 return title  
  
 def clean\_content(self):  
 content = self.cleaned\_data.get('content')  
 if not content:  
 raise forms.ValidationError("内容不能为空！")  
 if len(content) < 10:  
 raise forms.ValidationError("内容长度不能少于 10 个字符！")  
 return content

**#新建文件templates/ registration/login.html**

{% extends 'base.html' %}  
  
{% block content %}  
 <h1 class="title is-1">登录</h1>  
 <form method="post">  
 {% csrf\_token %}  
 <div class="field">  
 <label class="label">用户名</label>  
 <div class="control">  
 <input class="input" type="text" name="username" required>  
 </div>  
 </div>  
 <div class="field">  
 <label class="label">密码</label>  
 <div class="control">  
 <input class="input" type="password" name="password" required>  
 </div>  
 </div>  
 <div class="field">  
 <div class="control">  
 <button type="submit" class="button is-primary">登录</button>  
 </div>  
 </div>  
 </form>  
{% endblock %}

templates/新建以下模板文件

**#403.html**

{% extends 'base.html' %}  
  
{% block content %}  
 <div class="notification is-danger">  
 <h1 class="title is-1">403 禁止访问</h1>  
 <p>你没有权限执行此操作。</p>  
 <a href="{% url 'post\_list' %}" class="button is-link">返回首页</a>  
 </div>  
{% endblock %}

**#base.html**

<!DOCTYPE html>  
<html lang="zh-CN">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>MyBlog</title>  
 <!-- Bulma CSS -->  
 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.4/css/bulma.min.css">  
</head>  
<body>  
<!-- 导航栏 -->  
<nav class="navbar is-light" role="navigation" aria-label="main navigation">  
 <div class="navbar-brand">  
 <a class="navbar-item" href="{% url 'post\_list' %}">  
 <strong>MyBlog</strong>  
 </a>  
 </div>  
  
 <div class="navbar-menu">  
 <div class="navbar-end">  
 <div class="navbar-item">  
 <div class="buttons">  
 {% if user.is\_authenticated %}  
 <a class="button is-primary" href="{% url 'post\_create' %}">  
 <strong>新建文章</strong>  
 </a>  
 <!-- 将登出按钮改为表单 -->  
 <form action="{% url 'logout' %}" method="post" style="display: inline;">  
 {% csrf\_token %}  
 <button type="submit" class="button is-light">退出</button>  
 </form>  
 {% else %}  
 <a class="button is-primary" href="{% url 'login' %}">  
 <strong>登录</strong>  
 </a>  
 {% endif %}  
 </div>  
 </div>  
 </div>  
 </div>  
</nav>  
  
 <!-- 消息提示 -->  
 {% if messages %}  
 <div class="container mt-4">  
 {% for message in messages %}  
 <div class="notification is-{{ message.tags }}">  
 {{ message }}  
 </div>  
 {% endfor %}  
 </div>  
 {% endif %}  
  
 <!-- 主要内容 -->  
 <section class="section">  
 <div class="container">  
 {% block content %}  
 {% endblock %}  
 </div>  
 </section>  
  
 <!-- 页脚 -->  
 <footer class="footer">  
 <div class="content has-text-centered">  
 <p>  
 <strong>MyBlog</strong> © 2023. Powered by Django and Bulma.  
 </p>  
 </div>  
 </footer>  
  
 <!-- Bulma JS -->  
 <script>  
 // 控制模态框的显示和隐藏  
 *document*.addEventListener('DOMContentLoaded', () => {  
 // 获取所有模态框关闭按钮  
 const closeButtons = *document*.querySelectorAll('.modal .delete, .modal .modal-background');  
 closeButtons.forEach(button => {  
 button.addEventListener('click', () => {  
 const modal = button.closest('.modal');  
 modal.classList.remove('is-active');  
 });  
 });  
 });  
 </script>  
</body>  
</html>

**#post\_create.html**

{% extends 'base.html' %}  
{% load static %}  
  
{% block content %}  
 <h1 class="title is-1">新建文章</h1>  
 <form method="post" id="article-form">  
 {% csrf\_token %}  
  
 <!-- 标题字段 -->  
 <div class="field">  
 <label class="label">标题</label>  
 <div class="control">  
 {{ form.title }}  
 </div>  
 {% if form.title.errors %}  
 <div class="help is-danger">  
 {% for error in form.title.errors %}  
 {{ error }}  
 {% endfor %}  
 </div>  
 {% endif %}  
 </div>  
  
 <!-- 内容字段 -->  
 <div class="field">  
 <label class="label">内容</label>  
 <div class="control">  
 <textarea id="editor" name="content"></textarea>  
 </div>  
 {% if form.content.errors %}  
 <div class="help is-danger">  
 {% for error in form.content.errors %}  
 {{ error }}  
 {% endfor %}  
 </div>  
 {% endif %}  
 </div>  
  
 <!-- 提交按钮 -->  
 <div class="field">  
 <div class="control">  
 <button type="submit" class="button is-primary">提交</button>  
 </div>  
 </div>  
 </form>  
  
 <!-- 引入 CKEditor 5 -->  
 <script src="https://cdn.ckeditor.com/ckeditor5/41.1.0/classic/ckeditor.js"></script>  
 <script>  
 *document*.addEventListener('DOMContentLoaded', () => {  
 let editor;  
  
 // 初始化 CKEditor 5  
 ClassicEditor  
 .create(*document*.querySelector('#editor'), {  
 // 自定义工具栏  
 toolbar: [  
 'heading', '|',  
 'bold', 'italic', 'underline', 'strikethrough', '|',  
 'link', 'bulletedList', 'numberedList', '|',  
 'blockQuote', 'undo', 'redo',  
 ],  
 // 其他配置  
 placeholder: '请输入内容...',  
 })  
 .then(instance => {  
 editor = instance; // 将编辑器实例保存到变量中  
 *console*.log('CKEditor 5 初始化成功', editor);  
 })  
 .catch(error => {  
 *console*.error('CKEditor 5 初始化失败', error);  
 });  
  
 // 表单提交时，同步 CKEditor 5 的内容到 <textarea>  
 const form = *document*.querySelector('#article-form');  
 form.addEventListener('submit', () => {  
 const editorData = editor.getData(); // 获取编辑器内容  
 *document*.querySelector('#editor').value = editorData; // 同步到 <textarea>  
 });  
 });  
 </script>  
{% endblock %}

**#post\_delete.html**

{% extends 'base.html' %}  
  
{% block content %}  
 <h1 class="title is-1">删除文章</h1>  
 <div class="content">  
 <p>你确定要删除文章 <strong>{{ post.title }}</strong> 吗？此操作不可撤销。</p>  
 </div>  
  
 <form method="post">  
 {% csrf\_token %}  
 <div class="buttons">  
 <button type="submit" class="button is-danger">确认删除</button>  
 <a href="{% url 'post\_detail' post.id %}" class="button">取消</a>  
 </div>  
 </form>  
{% endblock %}

**#post\_detail.html**

{% extends 'base.html' %}  
  
{% block content %}  
 <h1 class="title is-1">{{ post.title }}</h1>  
 <p class="subtitle is-6">作者：{{ post.author.username }} | 发布于：{{ post.created\_at|date:"Y-m-d H:i" }} | 最后更新：{{ post.updated\_at|date:"Y-m-d H:i" }}</p>  
 <div class="content">  
 {{ post.content }}  
 </div>  
  
 {% if user == post.author %}  
 <div class="buttons mt-5">  
 <a href="{% url 'post\_update' post.id %}" class="button is-primary">编辑</a>  
 <a href="{% url 'post\_delete' post.id %}" class="button is-danger">删除</a>  
 </div>  
 {% endif %}  
{% endblock %}

**#post\_list.html**

{% extends 'base.html' %}  
  
{% block content %}  
 <h1 class="title is-1">文章列表</h1>  
 {% for post in page\_obj %}  
 <div class="card mb-5">  
 <div class="card-content">  
 <div class="media">  
 <div class="media-content">  
 <p class="title is-4">{{ post.title }}</p>  
 <p class="subtitle is-6">作者：{{ post.author.username }} | 发布于：{{ post.created\_at|date:"Y-m-d H:i" }}</p>  
 </div>  
 </div>  
 <div class="content">  
 {{ post.content|truncatewords:30 }}  
 <br>  
 <a href="{% url 'post\_detail' post.id %}" class="button is-link is-small">阅读全文</a>  
 </div>  
 </div>  
 </div>  
 {% endfor %}  
  
 <!-- 分页 -->  
 <nav class="pagination is-centered" role="navigation" aria-label="pagination">  
 {% if page\_obj.has\_previous %}  
 <a href="?page={{ page\_obj.previous\_page\_number }}" class="pagination-previous">上一页</a>  
 {% endif %}  
 {% if page\_obj.has\_next %}  
 <a href="?page={{ page\_obj.next\_page\_number }}" class="pagination-next">下一页</a>  
 {% endif %}  
  
 <ul class="pagination-list">  
 {% for num in page\_obj.paginator.page\_range %}  
 <li>  
 <a href="?page={{ num }}" class="pagination-link {% if page\_obj.number == num %}is-current{% endif %}">  
 {{ num }}  
 </a>  
 </li>  
 {% endfor %}  
 </ul>  
 </nav>  
{% endblock %}

**#post\_update.html**

{% extends 'base.html' %}  
{% load static %}  
  
{% block content %}  
 <h1 class="title is-1">编辑文章</h1>  
 <form method="post" id="article-form">  
 {% csrf\_token %}  
  
 <!-- 标题字段 -->  
 <div class="field">  
 <label class="label">标题</label>  
 <div class="control">  
 {{ form.title }}  
 </div>  
 {% if form.title.errors %}  
 <div class="help is-danger">  
 {% for error in form.title.errors %}  
 {{ error }}  
 {% endfor %}  
 </div>  
 {% endif %}  
 </div>  
  
 <!-- 内容字段 -->  
 <div class="field">  
 <label class="label">内容</label>  
 <div class="control">  
 <textarea id="editor" name="content">{{ form.content.value|default\_if\_none:'' }}</textarea>  
 </div>  
 {% if form.content.errors %}  
 <div class="help is-danger">  
 {% for error in form.content.errors %}  
 {{ error }}  
 {% endfor %}  
 </div>  
 {% endif %}  
 </div>  
  
 <!-- 提交按钮 -->  
 <div class="field">  
 <div class="control">  
 <button type="submit" class="button is-primary">更新</button>  
 </div>  
 </div>  
 </form>  
  
 <!-- 引入 CKEditor 5 -->  
 <script src="https://cdn.ckeditor.com/ckeditor5/41.1.0/classic/ckeditor.js"></script>  
 <script>  
 *document*.addEventListener('DOMContentLoaded', () => {  
 let editor;  
  
 // 初始化 CKEditor 5  
 ClassicEditor  
 .create(*document*.querySelector('#editor'), {  
 // 自定义工具栏  
 toolbar: [  
 'heading', '|',  
 'bold', 'italic', 'underline', 'strikethrough', '|',  
 'link', 'bulletedList', 'numberedList', '|',  
 'blockQuote', 'undo', 'redo'  
 ],  
 // 其他配置  
 placeholder: '请输入内容...',  
 })  
 .then(instance => {  
 editor = instance; // 将编辑器实例保存到变量中  
 *console*.log('CKEditor 5 初始化成功', editor);  
 })  
 .catch(error => {  
 *console*.error('CKEditor 5 初始化失败', error);  
 });  
  
 // 表单提交时，同步 CKEditor 5 的内容到 <textarea>  
 const form = *document*.querySelector('#article-form');  
 form.addEventListener('submit', () => {  
 const editorData = editor.getData(); // 获取编辑器内容  
 *document*.querySelector('#editor').value = editorData; // 同步到 <textarea>  
 });  
 });  
 </script>  
{% endblock %}

**#user\_posts.html**

{% extends 'base.html' %}  
  
{% block content %}  
 <h1 class="title is-1">我的文章</h1>  
 {% for post in user\_posts %}  
 <div class="card mb-5">  
 <div class="card-content">  
 <div class="media">  
 <div class="media-content">  
 <p class="title is-4">{{ post.title }}</p>  
 <p class="subtitle is-6">发布于：{{ post.created\_at|date:"Y-m-d H:i" }}</p>  
 </div>  
 </div>  
 <div class="content">  
 {{ post.content|truncatewords:30 }}  
 <br>  
 <a href="{% url 'post\_detail' post.id %}" class="button is-link is-small">阅读全文</a>  
 </div>  
 </div>  
 </div>  
 {% endfor %}  
{% endblock %}

**迁移数据库**

python manage.py migrate

**创建超级用户**

python manage.py createsuperuser

**运行服务器**

python manage.py runserver

**一、项目背景**

使用 Python 的 Django 框架搭建一个功能完备的个人博客系统，实现文章的创建、读取、更新与删除操作，以及基本的用户认证和权限管理，让博主能够便捷地管理自己的博客内容。

**二、项目要求**

1. **模型设计**

创建 Post（文章）模型，包含以下字段：

* title：文章标题，字符串类型，最大长度为 200，不能为空。
* content：文章内容，文本类型。
* author：文章作者，关联到 Django 内置的 User 模型，表明文章所属用户，级联删除。
* created\_at：文章创建时间，自动设置为当前时间。
* updated\_at：文章更新时间，每次更新文章时自动更新为当前时间。
* 创建合适的模型管理器，用于自定义查询方法（如获取某个用户的所有文章）。

1. **视图与 URL 设计**

设计以下视图函数及对应的 URL 路由：

* post\_list：展示所有文章的列表，按照创建时间倒序排列，每页显示 5 篇文章，需要实现分页功能。访问 URL：/blog/
* post\_detail：显示单篇文章的详细内容，根据文章的 id 进行查询展示。访问 URL：/blog/<int:post\_id>/
* post\_create：提供文章创建的表单页面，仅登录用户可访问，提交表单后创建新文章并跳转到文章详情页。访问 URL：/blog/create/
* post\_update：已登录用户可对自己的文章进行编辑，进入编辑页面（预填充原文章内容），提交后更新文章并跳转回文章详情页，非文章作者禁止访问。访问 URL：/blog/<int:post\_id>/update/
* post\_delete：文章作者可删除自己的文章，点击删除按钮后弹出确认框，确认后删除文章并重定向到文章列表页，非作者禁止访问。访问 URL：/blog/<int:post\_id>/delete/

1. **模板设计**

基于 Bootstrap 或其他前端框架设计简洁美观的模板：

* base.html：作为基础模板，包含网站的头部（导航栏、博客标题等）、底部（版权信息等）以及通用的 CSS 和 JavaScript 引入。
* post\_list.html：继承自 base.html，用于展示文章列表，每篇文章以卡片形式呈现，显示标题、作者、创建时间，提供查看全文的链接。
* post\_detail.html：继承自 base.html，展示单篇文章的完整内容，包括标题、作者、创建时间、更新时间，提供编辑和删除按钮（根据用户权限显示或隐藏）。
* post\_create.html 和 post\_update.html：继承自 base.html，包含用于创建和更新文章的表单，表单字段包括标题和内容，提交按钮。

1. **用户认证与权限管理**

使用 Django 内置的认证系统，确保只有登录用户才能创建、编辑和删除文章。

在视图函数中进行权限判断，对于无权限的操作给予友好的提示信息（如 403 Forbidden 页面，并显示相应错误文本）。

1. **数据验证与错误处理**

在创建和更新文章时，对用户输入的标题和内容进行前端和后端验证，确保标题不为空，内容长度符合合理范围，若有错误则在表单页面提示用户错误信息。