Sistema Blog Admin

Documentación Técnica Completa

Django 5.1.2
Sistema de Gestión de Blogs

Versión 1.0 - 4 de junio de 2025

Desarrollador: Eduardo Mellado
Tecnologia: Django 5.1.2 + Bootstrap 5
Base de datos: SQLite3
Frontend: HTML5, CSS3, JavaScript

Índice general

Índice de figuras

Índice de cuadros

0.1. Introducción

0.1.1. Descripción del Sistema

El **Sistema Blog Admin** es una aplicación web desarrollada en Django que permite a los usuarios crear, gestionar y publicar blogs de forma intuitiva. El sistema cuenta con una interfaz administrativa moderna y templates públicos personalizables.

0.1.2. Características Principales

- **Gestión completa de blogs:** CRUD (Crear, Leer, Actualizar, Eliminar)
- **Sistema de imágenes:** Subida y gestión de imágenes por blog
- **Múltiples diseños:** Templates públicos intercambiables
- **Control de publicación:** Estados de borrador y publicado
- **Autenticación robusta:** Sistema de usuarios integrado
- **Interfaz moderna:** Bootstrap 5 con modo oscuro
- **Responsive design:** Optimizado para dispositivos móviles

0.1.3. Tecnologías Utilizadas

Categoría	**Tecnología**
Backend	Django 5.1.2
Base de datos	SQLite3
Frontend	Bootstrap 5.3.3
Iconos	Bootstrap Icons 1.11.1
Formularios	Django Widget Tweaks
Lenguaje	Python 3.12+
Servidor web	Django Development Server

Cuadro 1: Stack tecnológico del sistema

0.2. Arquitectura del Sistema

0.2.1. Estructura General

El proyecto sigue el patrón MVT (Model-View-Template) de Django con una arquitectura modular que separa claramente las responsabilidades.

Estructura del Proyecto

```
proyecto_blogs/
|-- manage.py
|-- proyecto_blogs/          # Configuracion principal
|   |-- settings.py
|   |-- urls.py
|   |-- wsgi.py
|   +-- asgi.py
|-- blogs/                   # App principal
|   |-- models.py
|   |-- views.py
|   |-- forms.py
|   |-- urls.py
|   |-- admin.py
|   +-- templates/
|-- usuarios/                # App de autenticacion
|   |-- urls.py
|   +-- templates/
|-- templates/               # Templates base
|-- static/                  # Archivos estaticos
|-- media/                   # Archivos subidos
+-- db.sqlite3              # Base de datos
```

0.2.2. Diagrama de Arquitectura

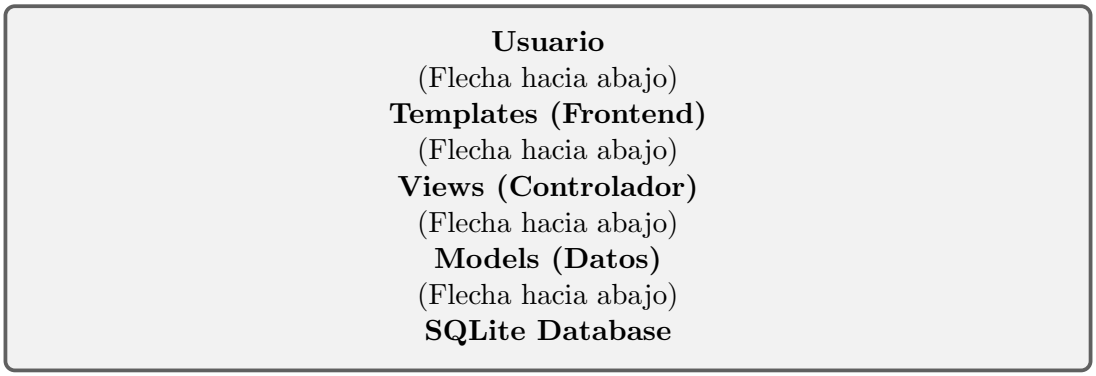

Figura 1: Flujo de datos del sistema MVT

0.3. Instalación y Configuración

0.3.1. Requisitos del Sistema

0.3.2. Instalación Paso a Paso

1. **Clonar o descargar el proyecto**

```
git clone <repositorio>
cd proyecto_blogs
```

2. **Crear entorno virtual**

Componente	Versión Mínima
Python	3.10+
pip	21.0+
Django	5.1.2
django-widget-tweaks	1.4.12

Cuadro 2: Requisitos del sistema

```
python -m venv venv
# Windows
venv\Scripts\activate
# Linux/Mac
source venv/bin/activate
```

3. **Instalar dependencias**

```
pip install django==5.1.2
pip install django-widget-tweaks
```

4. **Configurar base de datos**

```
python manage.py makemigrations
python manage.py migrate
```

5. **Crear superusuario**

```
python manage.py createsuperuser
```

6. **Ejecutar servidor de desarrollo**

```
python manage.py runserver
```

7. **Acceder al sistema**

   - Admin: http://127.0.0.1:8000/blogs/
   - Login: http://127.0.0.1:8000/usuarios/login/
   - Django Admin: http://127.0.0.1:8000/admin/

Capítulo 1

`models.py`

1.1. Introducción

El archivo `models.py` es uno de los componentes fundamentales en cualquier aplicación Django. Define los modelos de datos que representan las tablas en la base de datos. En este documento, analizaremos línea por línea el archivo `models.py` de la aplicación de blogs, explicando el propósito y funcionamiento de cada elemento.

1.2. Importaciones

```
from django.db import models
```

- **Propósito**: Importa el módulo `models` de Django, que contiene todas las clases base necesarias para crear modelos de base de datos.
- **Funcionalidad**: Proporciona acceso a:
  - La clase base `Model` de la que heredan todos los modelos
  - Tipos de campos como `CharField`, `TextField`, `DateTimeField`, etc.
  - Opciones de relaciones como `ForeignKey`, `ManyToManyField`, etc.

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

- **Propósito**: Importa el modelo `User` predefinido de Django.
- **Funcionalidad**:
  - Permite utilizar el sistema de autenticación integrado de Django
  - El modelo `User` incluye campos como username, password, email, first_name, last_name, etc.
  - Se utiliza para asociar blogs con usuarios específicos del sistema

```
import os
```

- **Propósito**: Importa el módulo `os` de Python estándar.
- **Funcionalidad**:
  - Proporciona funciones para interactuar con el sistema operativo
  - En este contexto, se usa específicamente `os.path.join()` para construir rutas de archivos de manera segura y portable

1.3. Función de Renombrado de Imágenes

```
def renombrar_imagen(instance, filename):
```

- **Propósito**: Define una función personalizada para renombrar archivos de imagen al subirlos.
- **Parámetros**:
  - `instance`: La instancia del modelo `Imagen` que se está guardando
  - `filename`: El nombre original del archivo subido
- **Uso**: Esta función se pasa como argumento al parámetro `upload_to` del campo `ImageField`

```
    blog_id = instance.blog.id or 'temp'
```

- **Propósito**: Obtiene el ID del blog asociado o asigna 'temp' si no existe.
- **Funcionalidad**:
  - `instance.blog` accede a la relación ForeignKey hacia el modelo Blog
  - `.id` obtiene el identificador único del blog
  - El operador `or` proporciona un valor por defecto si el blog aún no tiene ID (caso de creación simultánea)

```
    # Contar cuantas imagenes ya existen del blog
    total = Imagen.objects.filter(blog=instance.blog).count() + 1
```

- **Propósito**: Calcula el número secuencial para la nueva imagen.
- **Funcionalidad**:
  - `Imagen.objects` accede al manager del modelo Imagen
  - `.filter(blog=instance.blog)` filtra solo las imágenes del blog actual
  - `.count()` cuenta cuántas imágenes existen
  - Se suma 1 para obtener el número de la nueva imagen

```
    extension = filename.split('.')[-1]
```

- **Propósito**: Extrae la extensión del archivo original.
- **Funcionalidad**:
  - `split('.')` divide el nombre del archivo por puntos
  - `[-1]` obtiene el último elemento (la extensión)
  - Ejemplo: ïmagen.jpg"→ [ïmagen", "jpg"] → "jpg"

```
    nuevo_nombre = f"blog{blog_id}_{total:03d}.{extension}"
```

- **Propósito**: Construye el nuevo nombre del archivo usando f-strings.

- **Formato resultante**: blog[ID]_[NUMERO].extension
- **Detalles**:
  - El formato {total:03d} formatea el número con 3 dígitos, rellenando con ceros
  - Ejemplos: blog5_001.png, blog5_002.jpg, blog5_003.gif

```
    return os.path.join('imagenes_blogs', nuevo_nombre)
```

- **Propósito**: Retorna la ruta completa donde se guardará el archivo.
- **Funcionalidad**:
  - `os.path.join()` construye rutas de manera segura según el sistema operativo
  - La imagen se guardará en: MEDIA_ROOT/imagenes_blogs/blog[ID]_[NUMERO].extension

1.3.1. Modelo Blog

El modelo principal que almacena la información de cada blog.

```
class Blog(models.Model):
    propietario = models.ForeignKey(User, on_delete=models.CASCADE)
    titulo = models.CharField(max_length=255)
    descripcion = models.TextField(blank=True)
    fecha_creacion = models.DateTimeField(auto_now_add=True)
    publicar = models.BooleanField(default=False)
    seccion = models.CharField(max_length=100, default='blog1')

def __str__(self):
    return self.titulo
```

1.3.2. Modelo Imagen

Gestiona las imágenes asociadas a cada blog con renombrado automático.

```
def renombrar_imagen(instance, filename):
    blog_id = instance.blog.id or 'temp'
    total = Imagen.objects.filter(blog=instance.blog).count() + 1
    extension = filename.split('.')[-1]
    nuevo_nombre = f"blog{blog_id}_{total:03d}.{extension}"
    return os.path.join('imagenes_blogs', nuevo_nombre)

class Imagen(models.Model):
    blog = models.ForeignKey(Blog, on_delete=models.CASCADE,
    related_name='imagenes')
    imagen = models.ImageField(upload_to=renombrar_imagen)

    def __str__(self):
        return f"Imagen de {self.blog.titulo}"
```

1.3.3. Relaciones de Base de Datos

1.4. Sistema de Vistas

1.4.1. Vistas Principales

1.4.2. Decoradores de Seguridad

Todas las vistas administrativas utilizan el decorador `@login_required` para garantizar que solo usuarios autenticados puedan acceder.

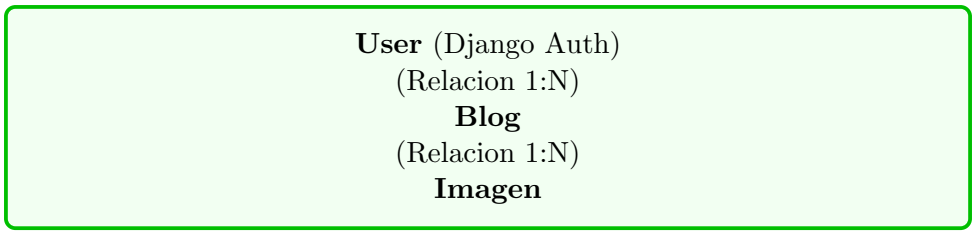

Figura 1.1: Diagrama de relaciones de la base de datos

Vista	URL	Función
lista_blogs	/blogs/	Dashboard principal
crear_blog	/blogs/crear/	Formulario de creación
editar_blog	/blogs/editar/<id>/	Edición de blog
eliminar_blog	/blogs/eliminar/<id>/	Confirmación de eliminación
subir_imagen	/blogs/subir-imagen/<id>/	Upload de imágenes
vista_cliente_publica	/blogs/<user>/<seccion>/	Vista pública

Cuadro 1.1: Mapeo de vistas principales

```
@login_required
def lista_blogs(request):
blogs = Blog.objects.filter(propietario=request.user)
return render(request, 'blogs/lista_blogs.html', {
  'blogs': blogs
})
```

1.5. Sistema de Templates

1.5.1. Jerarquía de Templates

El sistema utiliza herencia de templates para mantener consistencia visual.

Estructura de Templates

```
templates/
|-- base.html                 # Template base
|-- alejandro_blog1.html      # Diseno empresarial
|-- alejandro_blog2.html      # Diseno personal
+-- blogs/
|-- lista_blogs.html
|-- crear_blog.html
|-- editar_blog.html
|-- eliminar_blog.html
|-- subir_imagen.html
+-- eliminar_imagen.html
```

1.5.2. Features del Frontend

- **Bootstrap 5:** Framework CSS moderno
- **Modo oscuro:** `data-bs-theme="dark"`

- **Bootstrap Icons:** Iconografía consistente
- **Responsive design:** Adaptable a móviles
- **Validación en tiempo real:** JavaScript integrado
- **Loading states:** Feedback visual en formularios

1.6. Configuración Avanzada

1.6.1. Variables de Configuración

```
# Configuracion de archivos media
MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'media'

# Configuracion de autenticacion
LOGIN_URL = '/usuarios/login/'
LOGIN_REDIRECT_URL = '/blogs/'
LOGOUT_REDIRECT_URL = '/usuarios/login/'

# Localizacion
LANGUAGE_CODE = 'es-mx'
TIME_ZONE = 'America/Mexico_City'
USE_I18N = True
USE_L10N = True
USE_TZ = True
```

1.6.2. Configuración para Producción

```
# Para produccion, cambiar:
DEBUG = False
ALLOWED_HOSTS = ['tu-dominio.com', 'www.tu-dominio.com']

# Configurar base de datos PostgreSQL
DATABASES = {
  'default': {
    'ENGINE': 'django.db.backends.postgresql',
    'NAME': 'blog_db',
    'USER': 'blog_user',
    'PASSWORD': 'password',
    'HOST': 'localhost',
    'PORT': '5432',
  }
}

# Configurar archivos estaticos
STATIC_ROOT = '/var/www/static/'
STATICFILES_DIRS = [BASE_DIR / 'static']
```

1.7. Despliegue a Producción

1.7.1. Preparación del Servidor

1. **Instalar dependencias del sistema**

```
sudo apt install python3 python3-pip python3-venv
sudo apt install postgresql postgresql-contrib
sudo apt install nginx
```

2. **Configurar PostgreSQL**

```
sudo -u postgres psql
CREATE DATABASE blog_db;
CREATE USER blog_user WITH PASSWORD 'password';
GRANT ALL PRIVILEGES ON DATABASE blog_db TO blog_user;
\quit
```

3. **Configurar Nginx**

```
# Archivo: /etc/nginx/sites-available/blog_admin
server {
  listen 80;
  server_name tu-dominio.com;

  location / {
    proxy_pass http://127.0.0.1:8000;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
  }

  location /static/ {
    alias /var/www/static/;
  }

  location /media/ {
    alias /var/www/media/;
  }
}
```

1.7.2. Comandos de Despliegue

```
# Recopilar archivos estaticos
python manage.py collectstatic --noinput

# Aplicar migraciones
python manage.py migrate

# Ejecutar con Gunicorn
pip install gunicorn
gunicorn proyecto_blogs.wsgi:application --bind 127.0.0.1:8000
```

1.8. Mantenimiento y Extensiones

1.8.1. Backup de Base de Datos

```
# Backup
python manage.py dumpdata > backup.json

# Restaurar
python manage.py loaddata backup.json
```

1.8.2. Extensiones Futuras

- **Sistema de comentarios:** Interaccion con lectores
- **SEO avanzado:** Meta tags dinamicos
- **Analytics:** Estadisticas de visitas

- **API REST:** Para aplicaciones moviles
- **Multiidioma:** Soporte internacional

1.9. Solución de Problemas

1.9.1. Problemas Comunes

Problema	Solucion
Error 404 en login	Verificar `LOGIN_URL` en settings.py
Imagenes no se muestran	Configurar `MEDIA_URL` y `MEDIA_ROOT`
Error de permisos	Verificar decorador `@login_required`
CSS no carga	Ejecutar `collectstatic`

Cuadro 1.2: Problemas comunes y soluciones

1.10. Conclusion

El Sistema Blog Admin representa una solucion completa y profesional para la gestion de contenido web. Su arquitectura modular, interfaz intuitiva y codigo limpio lo convierten en una base solida para proyectos de blogging empresarial o personal.

La documentacion tecnica aqui presentada proporciona toda la informacion necesaria para instalar, configurar, mantener y extender el sistema segun las necesidades especificas del proyecto.

.1. Codigo de Configuracion Completo

.1.1. settings.py

```
from pathlib import Path
from django.utils.timezone import now

BASE_DIR = Path(__file__).resolve().parent.parent

SECRET_KEY = 'tu-clave-secreta-aqui'
DEBUG = True
ALLOWED_HOSTS = []

INSTALLED_APPS = [
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'widget_tweaks',
'blogs',
'usuarios',
]

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 = 'proyecto_blogs.urls'

    DATABASES = {
      'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
      }
    }

    LANGUAGE_CODE = 'es-mx'
    TIME_ZONE = 'America/Mexico_City'
    USE_I18N = True
    USE_L10N = True
    USE_TZ = True

    STATIC_URL = '/static/'
    STATICFILES_DIRS = [BASE_DIR / 'static']
    MEDIA_URL = '/media/'
    MEDIA_ROOT = BASE_DIR / 'media'

    LOGIN_URL = '/usuarios/login/'
    LOGIN_REDIRECT_URL = '/blogs/'
    LOGOUT_REDIRECT_URL = '/usuarios/login/'
```

.2. URLs Principal

```
    from django.contrib import admin
    from django.urls import path, include
    from django.conf import settings
    from django.conf.urls.static import static

    urlpatterns = [
    path('admin/', admin.site.urls),
    path('usuarios/', include('usuarios.urls')),
    path('blogs/', include('blogs.urls')),
    ]

    if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL,
    document_root=settings.MEDIA_ROOT)
```