

Manual de Django: Funciones CRUD



Ahora, ya que tenemos la plantilla completa, es momento de darle vida a nuestro blog creando las funciones de los CRUD en cada ventana.

Antes de iniciar, sabemos que el usuario debe de tener una foto de perfil, como no podemos modificar directamente la tabla User de django porque es mas complicado y riesgoso, podemos generar un modelo que sea una relación 1 a 1 entre la tabla User y la tabla archivo, para ello vamos a realizar el modelo de esta tabla en models.py

```
class Perfil(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    foto_perfil = models.ForeignKey(Archivo, null=True, blank=True,
on_delete=models.SET_NULL)

    def __str__(self):
        return f'Perfil de {self.user.username}'
```

¿Qué hace este modelo?

- Cada User tiene un perfil (OneToOneField)
- Cada perfil puede tener una foto, que es un archivo de la tabla Archivo
- Si se borra la imagen, no borra el perfil (por eso SET_NULL)

Crea un archivo core/signals.py:

```
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.contrib.auth.models import User
from .models import Perfil

@receiver(post_save, sender=User)
def crear_perfil_usuario(sender, instance, created, **kwargs):
    if created:
        Perfil.objects.create(user=instance)
```

Luego en tu apps.py agrega esto al final de tu class:

```
def ready(self):
    import core.signals
```

Ahora simplemente ejecuta la migración nuevamente:

```
(canacindra_env) C:\proyectoWeb>cd canacindra

(canacindra_env) C:\proyectoWeb\canacindra>python manage.py makemigrations
Migrations for 'core':
  core\migrations\0003_perfil.py
    + Create model Perfil

(canacindra_env) C:\proyectoWeb\canacindra>python manage.py migrate
System check identified some issues:

WARNINGS:
?: (mysql.W002) MySQL Strict Mode is not set for database connection 'default'
   HINT: MySQL's Strict Mode fixes many data integrity problems in MySQL, such as dat
ql-sql-mode
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, core, sessions
Running migrations:
  Applying core.0003_perfil... OK

(canacindra_env) C:\proyectoWeb\canacindra>
```

La tabla core_perfil ya debería ser visible en tu base de datos.

Usar @login_required en tus vistas

En views.py:

Importamos:

```
from django.contrib.auth.decorators import login_required
```

y a cada ventana que necesitemos volver privada, le agregamos lo siguiente en la parte superior:

```
@login_required
```

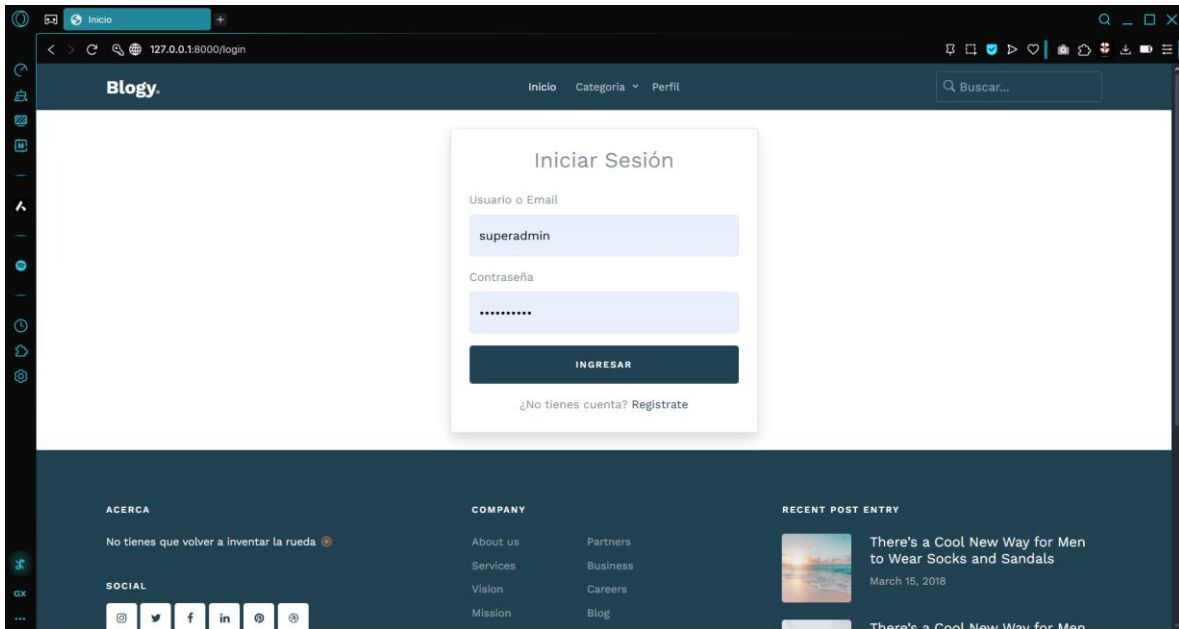
Ejemplo:

```
@login_required
def perfil (request):
    return render(request, 'core/perfil.html')

@login_required
def crud_perfil (request):
    return render(request, 'core/crud_perfil.html')
```

Con eso, si un usuario no está logueado y trata de acceder a una ventana privada, será redirigido automáticamente al login. Debemos hacer esto para las ventanas que creamos necesarias.

Ahora haremos CRUD de inicio de sesión:



Django ya trae todo lo necesario para autenticación: formularios, vistas, modelos de usuario, etc. Vamos a usar el sistema de autenticación que ya viene incluido para login, logout y registro.

Accedemos a settings.py y agregamos esto al final del archivo:

```
LOGIN_URL = 'core:login'
LOGIN_REDIRECT_URL = 'core:index'
LOGOUT_REDIRECT_URL = 'core:login'
```

Estas tres líneas en settings.py definen cómo Django maneja la autenticación (inicio y cierre de sesión).

LOGIN_URL = 'core:login'

Esto le dice a Django a dónde redirigir al usuario cuando intenta acceder a una vista protegida y no ha iniciado sesión.

Ejemplo:

- Si un usuario no autenticado visita una vista con `@login_required`, Django lo enviará a la URL llamada `'core:login'`.

Si no lo configuras, Django usa `/accounts/login/` por defecto.

LOGIN_REDIRECT_URL = 'core:index'

Esta es la URL a la que Django redirige al usuario después de iniciar sesión correctamente.

Ejemplo:

- Vas al login (`/login/`)
- Inicias sesión
- Te redirige automáticamente a la URL llamada `'core:index'`

Si no lo pones, Django redirige por defecto a `/accounts/profile/`.

LOGOUT_REDIRECT_URL = 'core:login'

Esto indica a dónde debe ir el usuario después de cerrar sesión.

Ejemplo:

- Haces clic en "Cerrar sesión"
- Django cierra la sesión
- Redirige a `'core:login'` (la página de login)

Si no lo configuras, Django simplemente muestra una página en blanco con un mensaje de "Logged out".

En `core:urls.py` debemos agregar lo siguiente:

```
from django.contrib.auth import views as auth_views
from django.contrib.auth.views import LogoutView

urlpatterns = [

    #A las ya existentes agregamos estas dos
```

```

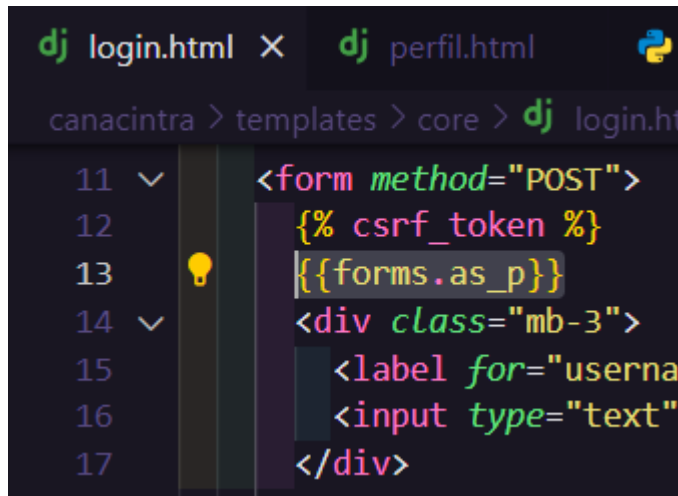
    path('login/',
auth_views.LoginView.as_view(template_name='core/login.html'),
name='login'),

    path('logout/', LogoutView.as_view(next_page='core:login'),
name='logout'),

]

```

En nuestra ventana de login agregamos:



```

11 <form method="POST">
12     {% csrf_token %}
13     {{ forms.as_p }}
14 <div class="mb-3">
15     <label for="username">
16     <input type="text"
17     </div>

```

Y en sidebar.html modificamos el botón de la siguiente manera:

```

<li>
    <form method="post" action="{% url 'core:logout' %}" class="w-100">
        {% csrf_token %}
        <button type="submit" class="btn btn-danger w-100 d-flex align-items-
center justify-content-center" data-bs-toggle="tooltip">
            <svg class="bi me-2" width="24" height="24"><use xlink:href= "#lock"/>
            </svg>
            <span>Cerrar Sesión</span>
        </button>
    </form>
</li>

```

Registrarse:

Paso 1: Vista sign_up en views.py

Agrega o edita esta vista:

```

def sign_up(request):
    if request.method == 'POST':

```

```

username = request.POST['username']
email = request.POST['email']
password1 = request.POST['password1']
password2 = request.POST['password2']

if password1 != password2:
    messages.error(request, 'Las contraseñas no coinciden.')
    return redirect('core:sign_up')

if User.objects.filter(username=username).exists():
    messages.error(request, 'El nombre de usuario ya está en uso.')
    return redirect('core:sign_up')

if User.objects.filter(email=email).exists():
    messages.error(request, 'El correo electrónico ya está
registrado.')
    return redirect('core:sign_up')

user = User.objects.create_user(username=username, email=email,
password=password1)
user.save()
messages.success(request, 'Tu cuenta ha sido creada con éxito. Ahora
puedes iniciar sesión.')
return redirect('core:login')

return render(request, 'core/sign_up.html')

```

Paso 2: URL en core/urls.py

```

path("sign_up", views.sign_up, name="sign_up"),

```

Paso 3: Mostrar mensajes en tu plantilla sign_up.html

Agrega esto justo debajo del <h3> para mostrar errores o confirmaciones:

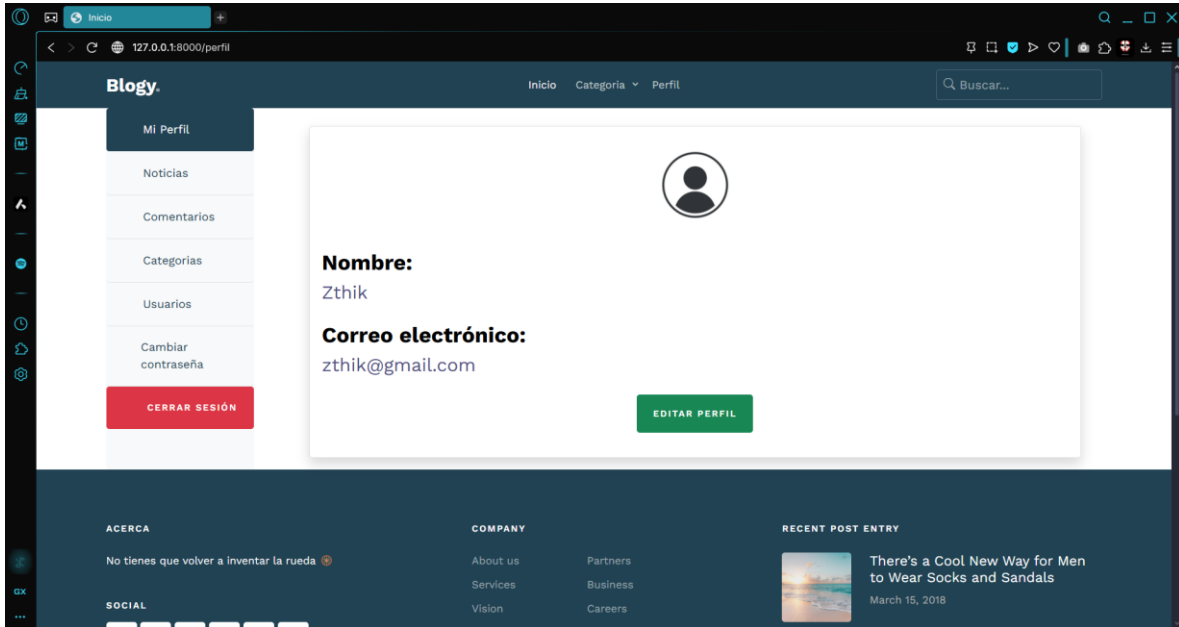
```

{% if messages %}
    {% for message in messages %}
        <div class="alert alert-{{ message.tags }}" alert-dismissible fade show"
role="alert">
            {{ message }}
            <button type="button" class="btn-close" data-bs-dismiss="alert" aria-
label="Cerrar"></button>
        </div>
    {% endfor %}
{% endif %}

```

```
{% endfor %}
{% endif %}
```

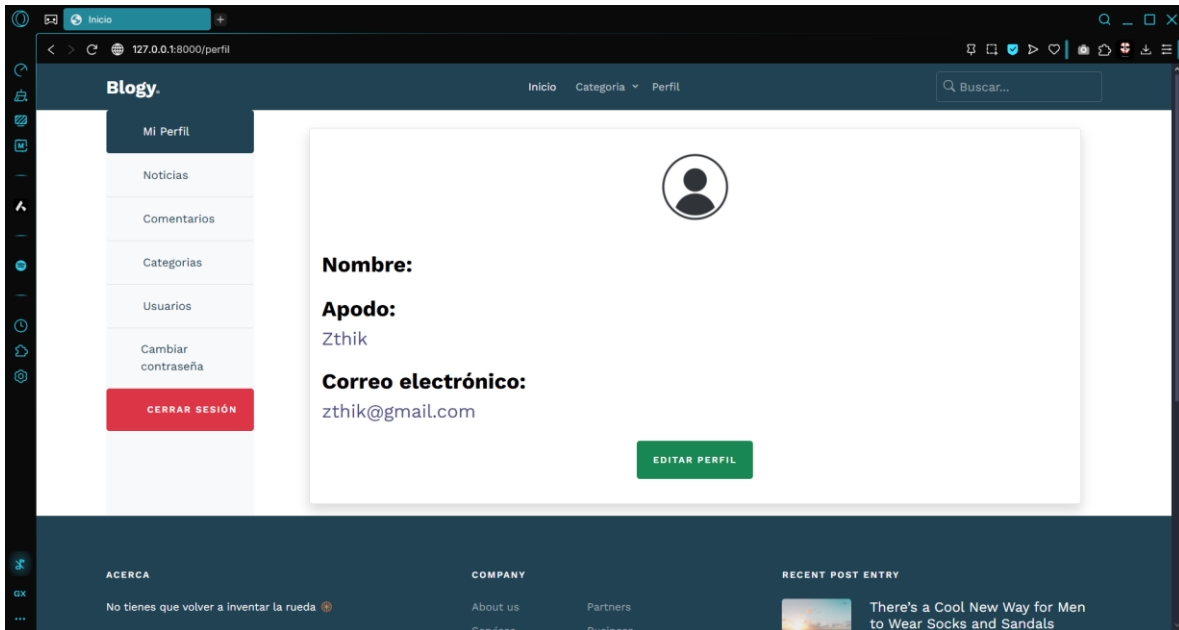
Esto debería ser suficiente para que puedas agregar usuarios usando el sistema de Django pero con tu propia ventana, prueba a agregar un nuevo usuario:



Puedes jugar con la ventana para mostrar mas datos:

```
<div class="mb-4 text-start">
  <h3 class="text-black fw-bold">Nombre:</h3>
  <h4 class="text-secondary mb-1">{{ user.name }}</h4>
</div>

<div class="mb-4 text-start">
  <h3 class="text-black fw-bold">Apodo:</h3>
  <h4 class="text-secondary mb-1">{{ user.username }}</h4>
</div>
```

Editar Perfil:

Configuración de medios en settings.py

Define dónde se guardan los archivos subidos

```
MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'media'
```

Configura en urls.py principal la entrega de archivos media en desarrollo:

```
from django.conf import settings
from django.conf.urls.static import static
urlpatterns = [
    # ... tus rutas ...
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Formulario HTML para editar perfil

Incluye campos para nombre, email y subir archivo de foto.

Usa enctype="multipart/form-data" en el <form> para subir archivos.

```
<form method="POST" enctype="multipart/form-data">
    {% csrf_token %}
```

```

<input type="text" name="nombre" value="{{ request.user.username }}" />
<input type="email" name="email" value="{{ request.user.email }}" />
<input type="file" name="foto" />
<button type="submit">Guardar Cambios</button>
</form>

```

Vista crud_perfil para actualizar perfil

Solo accesible para usuarios logueados (@login_required).

Actualiza nombre y email del usuario.

Si sube nueva foto, elimina la anterior (archivo físico y registro Archivo).

Guarda la nueva foto en Archivo y la relaciona en Perfil.

```

@login_required
def crud_perfil(request):
    user = request.user

    if request.method == 'POST':
        nombre = request.POST.get('nombre')
        email = request.POST.get('email')
        foto = request.FILES.get('foto')

        user.username = nombre
        user.email = email
        user.save()

        perfil, _ = Perfil.objects.get_or_create(user=user)

        if foto:
            if perfil.foto_perfil:

```

```

        archivo_anterior = perfil.foto_perfil

        ruta_fisica = os.path.join(settings.MEDIA_ROOT,
archivo_anterior.ruta)

        if os.path.exists(ruta_fisica):
            os.remove(ruta_fisica)
        archivo_anterior.delete()

    filename = default_storage.save(f"uploads/{foto.name}", foto)
    archivo = Archivo.objects.create(
        nombre=foto.name,
        nombre_temporal=filename,
        ruta=filename,
        tipo=foto.content_type,
        tamano=foto.size,
        fk_user=user
    )
    perfil.foto_perfil = archivo

    perfil.save()
    messages.success(request, 'Perfil actualizado correctamente.')
    return redirect('core:crud_perfil')

return render(request, 'core/crud_perfil.html')

```

Mostrar la foto de perfil en la plantilla

En el template, mostrar la foto si existe, si no mostrar foto por defecto:

```

{% if request.user.perfil.foto_perfil %}

{% else %}

```

```

{% endif %}
```

Con esto ya podemos iniciar y cerrar sesión, registrarnos e incluso editar nuestro perfil en nuestro blog, ahora vamos a tomar medidas en caso de que quieras cambiar tu contraseña:

1. Modifica el formulario para que envíe datos con POST y tenga csrf token:

```
<form method="POST">
  {% csrf_token %}
  <div class="mb-3">
    <label for="actual" class="form-label">Contraseña actual</label>
    <input type="password" class="form-control" id="actual" name="actual"
required>
  </div>

  <div class="mb-3">
    <label for="nueva" class="form-label">Nueva contraseña</label>
    <input type="password" class="form-control" id="nueva" name="nueva"
required>
  </div>

  <div class="mb-3">
    <label for="confirmar" class="form-label">Confirmar nueva
contraseña</label>
    <input type="password" class="form-control" id="confirmar"
name="confirmar" required>
  </div>

  <button type="submit" class="btn btn-primary w-100">Actualizar
Contraseña</button>
```

```
</form>
```

En views.py debemos modificar la vista para procesar el cambio, el código es el siguiente:

```
@login_required
def crud_cambContra(request):
    if request.method == 'POST':
        actual = request.POST.get('actual')
        nueva = request.POST.get('nueva')
        confirmar = request.POST.get('confirmar')

        user = request.user

        # Verificar contraseña actual
        if not user.check_password(actual):
            messages.error(request, 'La contraseña actual es incorrecta.')
            return redirect('core:crud_cambContra')

        # Verificar que las nuevas contraseñas coincidan
        if nueva != confirmar:
            messages.error(request, 'Las nuevas contraseñas no coinciden.')
            return redirect('core:crud_cambContra')

        # Cambiar contraseña
        user.set_password(nueva)
        user.save()

        # Para evitar que se cierre sesión al cambiar contraseña
        update_session_auth_hash(request, user)

        messages.success(request, 'Contraseña actualizada correctamente.')
        return redirect('core:crud_cambContra')

    return render(request, 'core/crud_cambContra.html')
```

4. Opcional: Muestra mensajes en el template

Al principio del archivo crud_cambContra.html agrega:

```
{% if messages %}
```

```

{% for message in messages %}

    <div class="alert alert-{{ message.tags }} alert-dismissible fade
show" role="alert">

        {{ message }}

        <button type="button" class="btn-close" data-bs-dismiss="alert"
aria-label="Cerrar"></button>

    </div>

{% endfor %}
{% endif %}

```

Con eso nuestra ventana para cambiar la contraseña ya debería funcionar correctamente.

Tabla de usuarios:

```

@login_required
def crud_usuarios(request):
    usuarios = User.objects.all().order_by('date_joined')
    return render(request, 'core/crud_usuarios.html', {'usuarios':
usuarios})

```

Creamos un bucle en el cuerpo de la tabla en nuestra ventana para imprimir los datos de los usuarios:

```

<tbody>
    {% for user in usuarios %}
    <tr>
        <td>{{ forloop.counter }}</td>
        <td>{{ user.get_full_name|default:user.username }}</td>
        <td>{{ user.email }}</td>
        <td>
            {% if user.is_superuser %}
            Administrador
            {% elif user.is_staff %}
            Editor
            {% else %}
            Colaborador

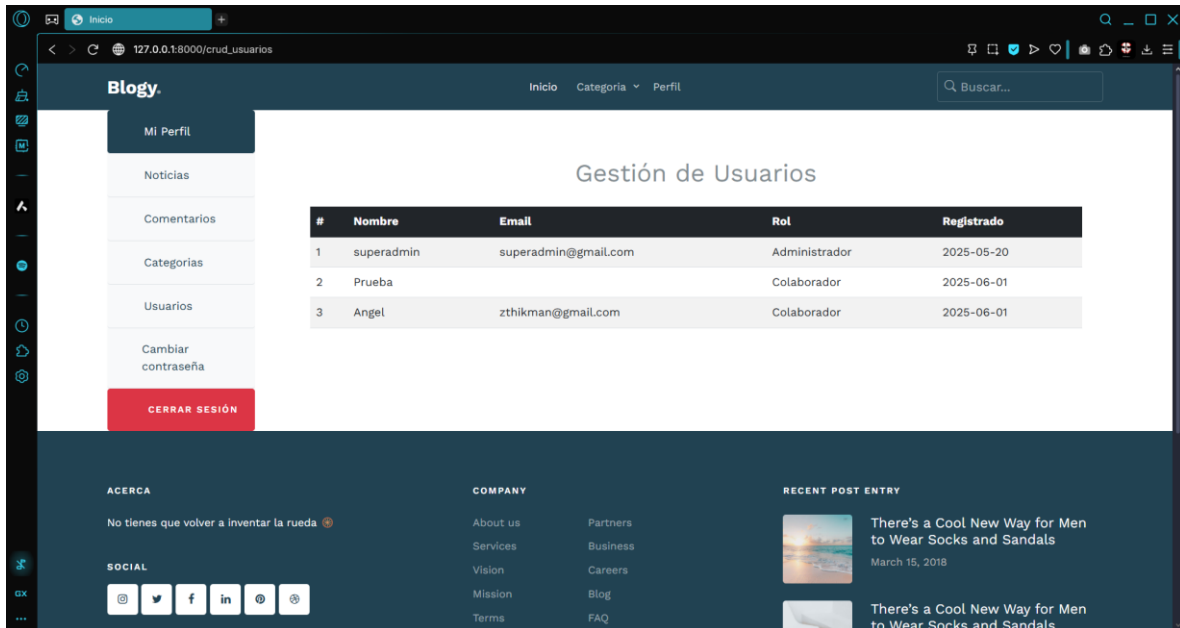
```

```

        {% endif %}
    </td>
    <td>{{ user.date_joined|date:"Y-m-d" }}</td>
</tr>
{% endfor %}
</tbody>

```

Y con esto ya podemos ver los usuarios que hay en nuestro blog:



Categorías:

Su funcionamiento es igual al de las demás ventanas, solo necesitamos cambiar rutas:

En la vista:

```

@login_required
def crud_categorias(request):
    if request.method == 'POST':
        nombre = request.POST.get('nombre')
        if nombre:
            Categoria.objects.create(
                nombre=nombre,
                fk_user=request.user,
                createdat=timezone.now()
            )
        return redirect('core:crud_categorias')

```

```

categorias = Categoria.objects.all().order_by('-createdat')
return render(request, 'core/crud_categorias.html', {
    'categorias': categorias
})

```

En su template:

```

<div class="container mt-5">
  <h2 class="text-center mb-4">Administrar Categorías</h2>

  <!-- Formulario para agregar nueva categoría -->
  <div class="card mb-4">
    <div class="card-body">
      <form method="POST" class="row g-3">
        {% csrf_token %}
        <div class="col-md-9">
          <input type="text" name="nombre" class="form-control"
placeholder="Nueva categoría" required>
        </div>
        <div class="col-md-3">
          <button type="submit" class="btn btn-primary w-
100">Agregar</button>
        </div>
      </form>
    </div>
  </div>

  <!-- Tabla de categorías -->
  <table class="table table-striped table-hover align-middle">
    <thead class="table-light">
      <tr>
        <th>#</th>
        <th>Nombre</th>
        <th>Registrado por</th>
        <th>Fecha</th>
      </tr>
    </thead>
    <tbody>
      {% for categoria in categorias %}
      <tr>
        <td>{{ forloop.counter }}</td>
        <td>{{ categoria.nombre }}</td>

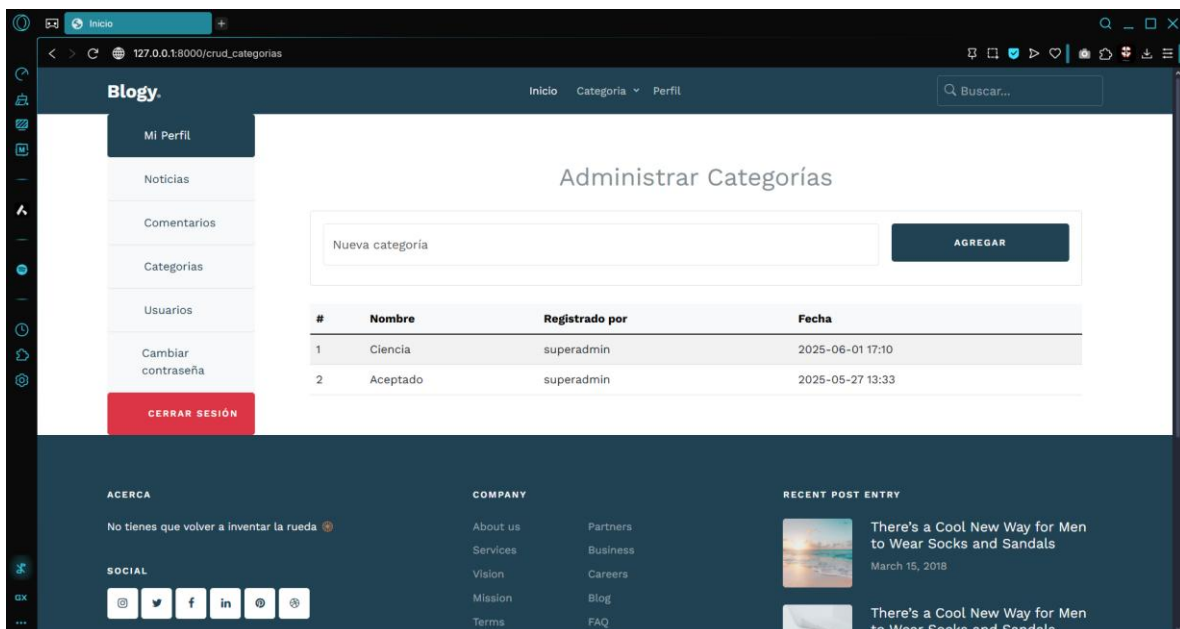
```



```

        <td>{{ categoria.fk_user.username }}</td>
        <td>{{ categoria.createdat|date:"Y-m-d H:i" }}</td>
    </tr>
    {% empty %}
    <tr>
        <td colspan="4" class="text-center">No hay categorías aún.</td>
    </tr>
    {% endfor %}
</tbody>
</table>
</div>

```



Esta misma lógica debe ser aplicada para realizar las ventanas faltantes, solo debes repetir los pasos, agregar, quitar o mover partes, la ventana de noticias se compone de todas estas ventanas, llamar imágenes, llamar valores de las tablas etc.