TUTORIALES

SEGURIDAD, CONTROL DE ACCESO Y ROLES DE USUARIO

Introducción

En cualquier aplicación web profesional, **la seguridad y el control de acceso** son fundamentales. No basta con mostrar u ocultar elementos visuales: hay que asegurarse de que el servidor **restrinja el acceso a rutas, datos y funcionalidades según el tipo de usuario** (rol) y su estado de autenticación.

Django incluye un sistema robusto de autenticación y autorización que permite:

- Verificar si un usuario está logueado.
- Asignar roles o permisos.
- Proteger vistas y rutas.
- Mostrar u ocultar contenido dinámico en función del rol.
- Gestionar errores de forma controlada y profesional.

Estos conceptos se aplican a lo largo de todo el desarrollo backend y frontend del proyecto.

1. Control de Acceso a Rutas y Vistas

¿Qué significa restringir una ruta?

Es limitar el acceso a ciertas páginas solo a determinados usuarios, por ejemplo:

- Que solo usuarios logueados puedan acceder al panel de usuario.
- Que solo administradores puedan entrar al panel de administración.

1.1 Usuarios Logueados: `@login_required`

Decorador que asegura que una vista solo sea accesible por usuarios autenticados.

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

@login_required
def panel_usuario(request):
    return render(request, 'usuario/panel.html')
```

🔒 Si el usuario no está logueado, es redirigido al `LOGIN_URL` definido en `settings.py`.

1.2 Control por Rol: `@user_passes_test`

Permite filtrar el acceso según una condición específica, como el rol del usuario.

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

def es_admin(user):
    return user.is_authenticated and user.role == 'admin'

@user_passes_test(es_admin)
def vista_admin(request):
    return render(request, 'admin/panel.html')
```

✅ Este decorador se usa cuando no es suficiente con estar logueado, sino que además se requiere un perfil concreto (admin, profesor, alumno, etc.).

---

2. Mostrar u Ocultar Elementos en Plantillas según el Usuario

2.1 Saber si el usuario está autenticado

```
{% if user.is_authenticated %}
  <p>Hola {{ user.username }}</p>
{% else %}
  <a href="{% url 'login' %}">Iniciar sesión</a>
{% endif %}
```

2.2 Mostrar contenido solo a ciertos roles

```
{% if user.role == 'admin' %}
  <a href="{% url 'panel_admin' %}">Administración</a>
{% endif %}
```

Importante: esto solo afecta al contenido visual. Nunca debe sustituir el control en la vista.

3. Páginas Genéricas de Error

Django permite definir vistas personalizadas para manejar errores como:

- `403` (prohibido)
- `404` (no encontrado)
- `500` (error interno)

3.1 Configuración en `settings.py`

```
LOGIN_URL = '/login/'  # Redirección automática si no está logueado

# Páginas de error
handler403 = 'mi_app.views.error_403'
handler404 = 'mi_app.views.error_404'
handler500 = 'mi_app.views.error_500'
```

3.2 Crear vistas de error en `views.py`

```
def error_403(request, exception=None):
    return render(request, 'errores/403.html', status=403)
```

Y su correspondiente template:

```
<!-- templates/errores/403.html>

<h1>403 - Acceso denegado</h1>
<p>No tienes permisos para ver esta página.</p>
```

4. Buenas Prácticas

- **Siempre** protege tus vistas desde el backend.
- Usa decoradores y mixins según el tipo de vista.
- No confíes solo en el frontend para ocultar contenido.
- Gestiona errores con páginas profesionales.
- Mantén centralizada la lógica de roles si es posible (funciones reutilizables o middleware).

Recomendación Avanzada

Para una lógica más potente y flexible puedes explorar estas librerías:

Librería	Función principal
`django-rules`	Permisos personalizados con funciones
`django-guardian`	Permisos a nivel de objeto