Django Auth: Seguridad, Usuarios y Permisos

Una guía completa para dominar la autenticación y autorización en Django, desde conceptos básicos hasta implementación práctica.

01

Comprender el modelo Auth

Exploraremos qué es el sistema de autenticación integrado de Django y cómo funciona.

02

Gestión de usuarios y sesiones

Aprenderemos a crear usuarios, manejar contraseñas de forma segura y gestionar sesiones.

03

Implementación práctica

Desarrollaremos funciones de login, logout y sistemas de permisos en aplicaciones reales.

¿Qué pasaría sin autenticación?

> **👉 ¿Qué pasaría si cualquier persona pudiera entrar a tu aplicación sin autenticación?**

Imagina un mundo digital sin puertas ni cerraduras. Tu aplicación web sería como una casa con todas las ventanas y puertas abiertas de par en par. Cualquier persona podría entrar, ver información confidencial, modificar datos críticos o incluso eliminar todo el contenido.

Idea clave

Sin autenticación, no hay seguridad. Django trae un sistema completo de auth listo para usar que protege tu aplicación desde el primer día.

¿Qué es el modelo Auth de Django?

El modelo Auth de Django es como tener un sistema de seguridad profesional integrado en tu aplicación. Es robusto, confiable y está listo para usar desde el momento en que creas tu proyecto.

Gestión de usuarios

Maneja usuarios y contraseñas de forma segura con encriptación automática.

Control de sesiones

Permite inicio y cierre de sesión con seguimiento automático del estado.

Sistema de permisos

Administra grupos y permisos granulares para controlar el acceso.

Todo esto se encuentra disponible en el paquete django.contrib.auth, que viene preinstalado con Django.

Componentes principales del sistema Auth

User (Usuario)

Representa a cada usuario individual en tu aplicación. Contiene información básica como username, email, nombre y apellido, además de metadatos de autenticación.

Permissions (Permisos)

Define acciones específicas que un usuario puede o no realizar. Por ejemplo: agregar posts, editar comentarios, eliminar usuarios.

Groups (Grupos)

Conjuntos de usuarios que comparten permisos comunes. Facilita la gestión masiva de autorizaciones como editores, moderadores, administradores.

Authentication

El proceso que verifica la identidad del usuario. Confirma que quien intenta acceder es realmente quien dice ser.

Autenticación vs Autorización

Aunque suenan similares, son conceptos fundamentalmente diferentes que trabajan en conjunto para proteger tu aplicación.

Autenticación

¿Quién eres?

Es el proceso de verificar la identidad de un usuario. Como mostrar tu cédula en la entrada de un edificio.

- **Login con usuario y contraseña**
- **Verificación biométrica**
- **Autenticación de dos factores**

Autorización

¿Qué puedes hacer?

Es el proceso de determinar qué acciones puede realizar un usuario autenticado. Como tener acceso solo a ciertos pisos del edificio.

- **Permisos de lectura/escritura**
- **Roles y grupos de usuarios**
- **Restricciones por área o función**

Seguridad integrada en Django

Django no solo maneja la autenticación, sino que incluye múltiples capas de protección que funcionan automáticamente para mantener tu aplicación segura.

Protección CSRF

Cross-Site Request Forgery protection evita que formularios maliciosos ejecuten acciones en nombre de usuarios legítimos.

Protección XSS

Cross-Site Scripting protection previene la inyección de scripts maliciosos en páginas web, escapando automáticamente el contenido dinámico.

Protección SQL Injection

El ORM de Django parametriza automáticamente las consultas, evitando que código malicioso se ejecute en la base de datos.

Hashing de contraseñas

Las contraseñas nunca se almacenan en texto plano. Django utiliza algoritmos criptográficos robustos para protegerlas.

Encriptación de contraseñas

Django implementa un sistema sofisticado de encriptación que garantiza que las contraseñas estén siempre protegidas, incluso si alguien accede directamente a la base de datos.

```
# settings.py
PASSWORD_HASHERS = [
    'django.contrib.auth.hashers.PBKDF2PasswordHasher',
    'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',
    'django.contrib.auth.hashers.Argon2PasswordHasher',
]
```

1. **Contraseña original**
   "mi_contraseña_123"
2. **Proceso de hash**
   PBKDF2 + Salt + Iteraciones
3. **Hash almacenado**
   pbkdf2_sha256$150000$...

👉 **Principio fundamental: Django nunca almacena contraseñas en texto claro. Cada contraseña es procesada por algoritmos criptográficos que la convierten en un hash irreversible.**

Creación de usuarios

Django proporciona métodos integrados para crear diferentes tipos de usuarios de manera segura y eficiente.

Ejemplo práctico en la shell de Django:

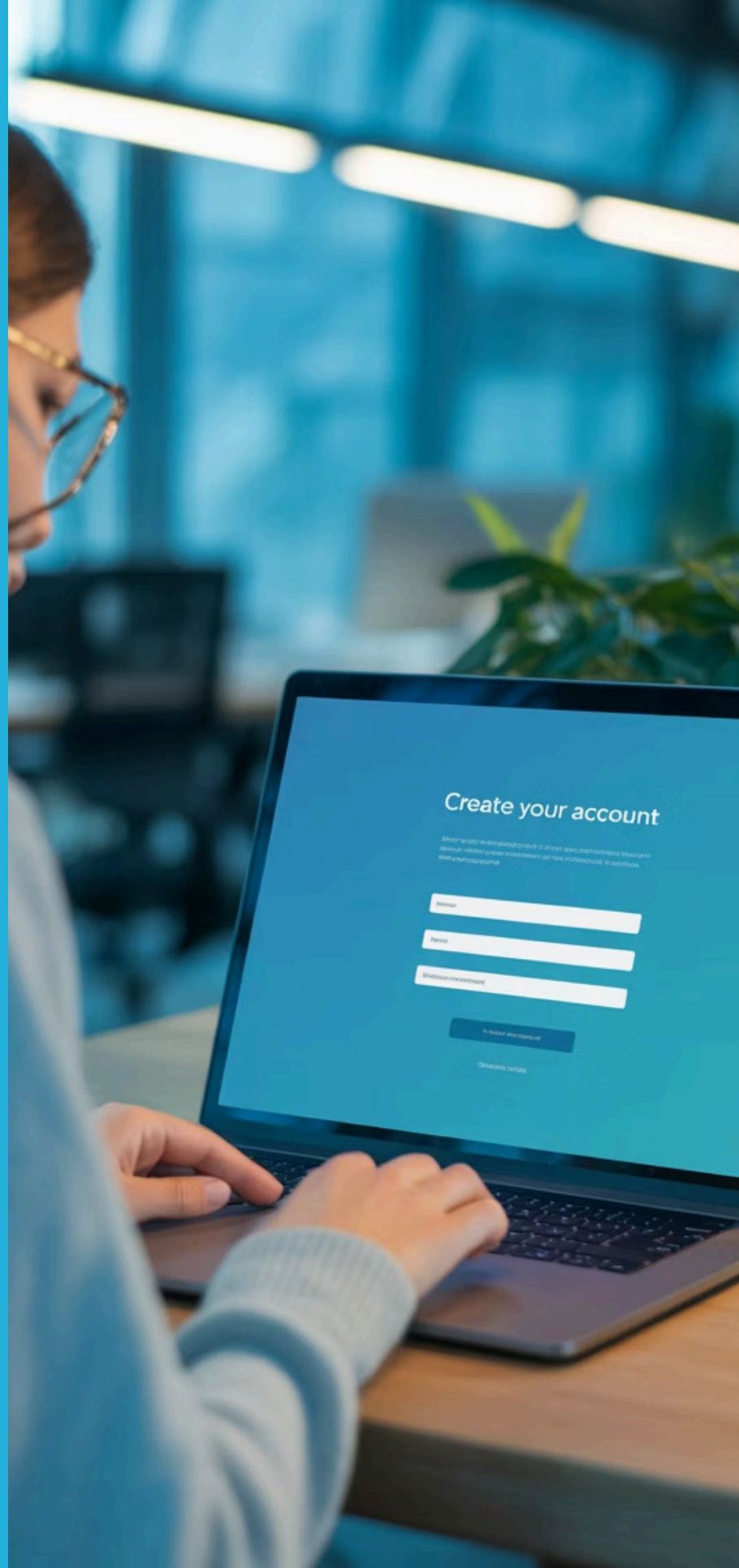
```
from django.contrib.auth.models import User

# Crear un usuario común
user = User.objects.create_user(
    username="juan",
    email="juan@ejemplo.com",
    password="contraseña_segura",
    first_name="Juan",
    last_name="Pérez"
)

# Crear un superusuario
admin = User.objects.create_superuser(
    username="admin",
    email="admin@miapp.com",
    password="admin123"
)
```

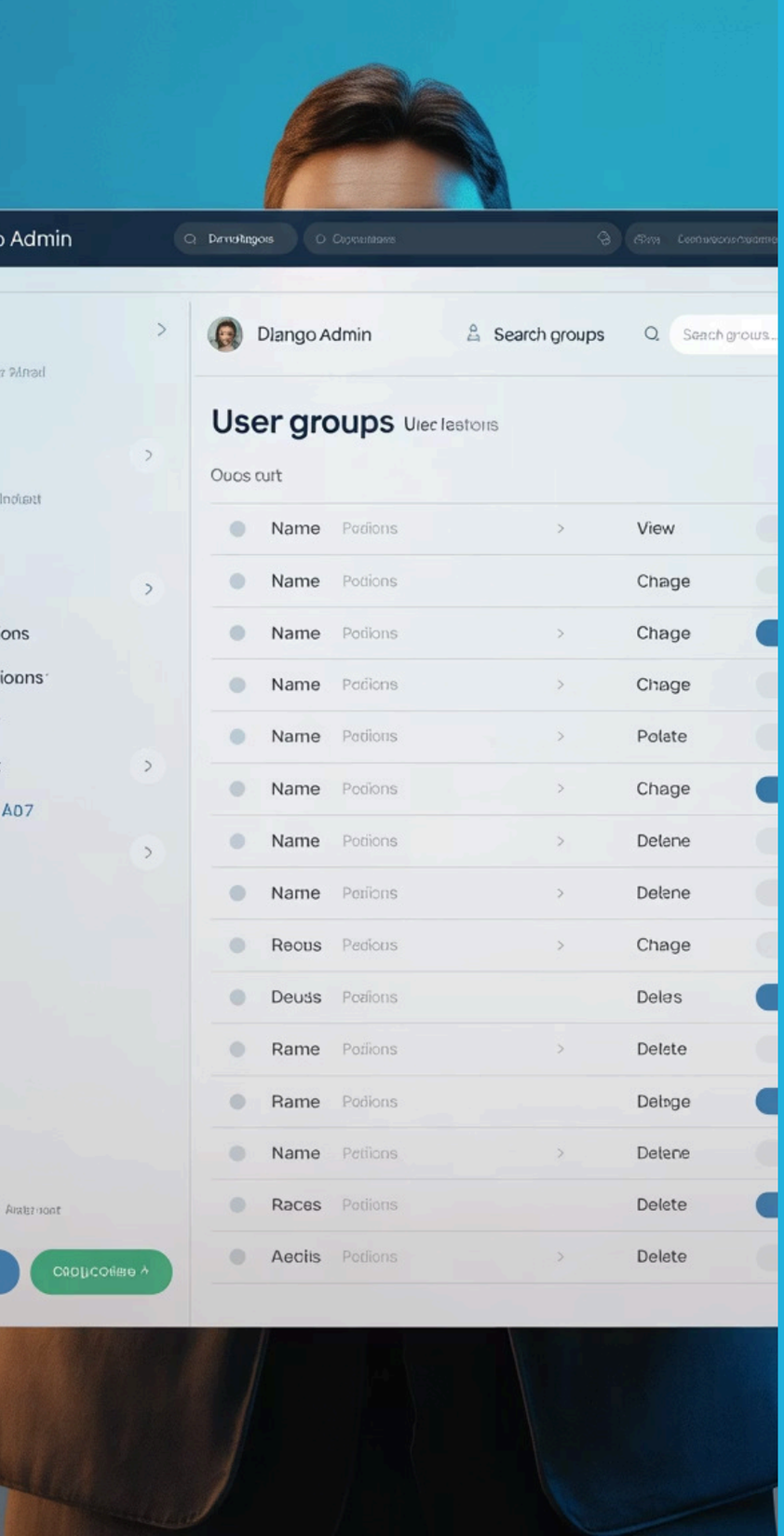
create_user()

Para usuarios regulares con permisos limitados. Perfecto para clientes, suscriptores o usuarios básicos.

create_superuser()

Para administradores con acceso completo. Incluye permisos para el panel administrativo de Django.





Actividad práctica 1

💻 Actividad en parejas

01

Crear superusuario

Ejecuta en la terminal de tu proyecto:

```
python manage.py createsuperuser
```

02

Acceder al Django Admin

Inicia tu servidor y visita `http://localhost:8000/admin/`

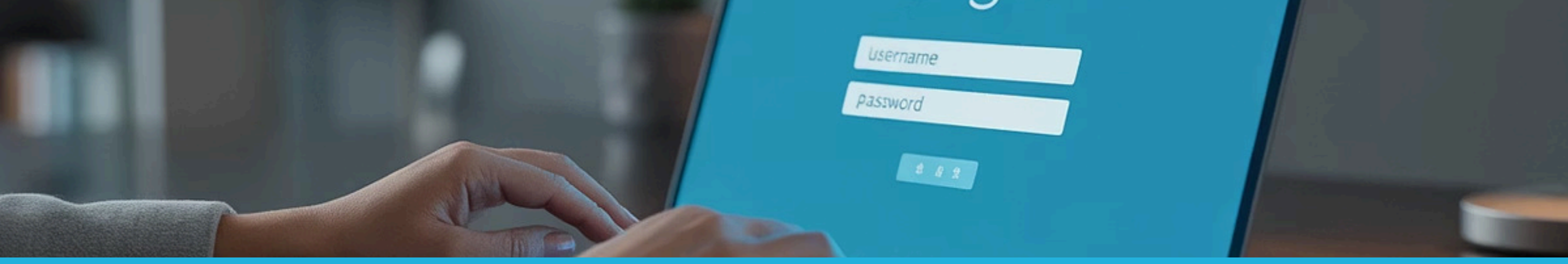
03

Explorar el sistema

Navega por las secciones de Users y Groups. Observa cómo Django organiza la información.

> **Objetivo: Familiarizarte con la interfaz administrativa de Django y comprender cómo se visualizan los usuarios y permisos en el sistema.**

Esta actividad te permitirá ver de primera mano cómo Django organiza y presenta la información de usuarios de manera intuitiva y profesional.



Autenticación de usuarios

El proceso de login es fundamental en cualquier aplicación. Django simplifica este proceso con funciones integradas que manejan la verificación y gestión de sesiones automáticamente.

Ejemplo de vista de login:

```
from django.contrib.auth import authenticate, login
from django.shortcuts import render, redirect
from django.contrib import messages

def login_view(request):
    if request.method == "POST":
        username = request.POST["username"]
        password = request.POST["password"]

        user = authenticate(
            request,
            username=username,
            password=password
        )

        if user is not None:
            login(request, user)
            messages.success(request, f"¡Bienvenido {user.first_name}!")
            return redirect("dashboard")
        else:
            messages.error(request, "Credenciales inválidas")

    return render(request, "login.html")
```

1 Recepción de datos

El formulario envía username y password

2 Verificación

authenticate() verifica las credenciales

3 Inicio de sesión

login() crea la sesión del usuario

Cierre de sesión (Logout)

El logout es tan importante como el login. Django proporciona una función simple pero poderosa que limpia toda la información de sesión y protege contra accesos no autorizados.

```
from django.contrib.auth import logout
from django.shortcuts import redirect
from django.contrib import messages

def logout_view(request):
    # Obtener el nombre del usuario antes del logout
    username = request.user.username

    # Cerrar la sesión
    logout(request)

    # Mostrar mensaje de despedida
    messages.info(request, f"¡Hasta pronto, {username}!")

    return redirect("login")
```

¿Qué hace logout()?

Borra la sesión actual del usuario, elimina cookies de autenticación y protege contra accesos indebidos posteriores.

👉 **Buena práctica: Siempre redirige a una página pública después del logout para confirmar que la sesión se cerró correctamente.**

El objeto request.user

Django automáticamente adjunta información del usuario a cada request, lo que te permite acceder fácilmente al estado de autenticación en cualquier vista o plantilla.

```
def mi_vista(request):
    if request.user.is_authenticated:
        print(f"Usuario logueado: {request.user.username}")
        print(f"Email: {request.user.email}")
        print(f"Es superusuario: {request.user.is_superuser}")
        print(f"Es staff: {request.user.is_staff}")
    else:
        print("Usuario anónimo")
```

1

is_authenticated

Verifica si el usuario ha iniciado sesión correctamente

2

is_superuser

Indica si tiene permisos de administrador completo

3

is_staff

Determina si puede acceder al panel administrativo

4

is_active

Confirma si la cuenta del usuario está activa

Estas propiedades están disponibles tanto en las vistas de Python como en las plantillas HTML, facilitando la personalización del contenido según el usuario.

Restricción de vistas

Django ofrece decoradores elegantes que permiten restringir el acceso a ciertas vistas solo a usuarios autenticados, simplificando enormemente la gestión de permisos.

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

@login_required
def perfil(request):
    return render(request, "perfil.html", {
        'usuario': request.user
    })

@login_required
def dashboard(request):
    return render(request, "dashboard.html")

# También puedes especificar URL de redirección
@login_required(login_url='/mi-login/')
def vista_privada(request):
    return render(request, "privada.html")
```

¿Cómo funciona @login_required?

Si el usuario no está autenticado, Django automáticamente lo redirige a la página de login. Una vez que inicie sesión exitosamente, será redirigido de vuelta a la vista original que intentaba acceder.

Configuración en settings.py

```
LOGIN_URL = '/accounts/login/'
LOGIN_REDIRECT_URL = '/dashboard/'
```

👉 **Resultado: Solo usuarios logueados pueden acceder. Los no autenticados son redirigidos automáticamente al login.**

Migraciones básicas

Las migraciones son el mecanismo que Django utiliza para aplicar cambios en la estructura de la base de datos de manera controlada y reversible.

Cambios en modelos

Modificas tus modelos Python

Crear migración

makemigrations genera el archivo de migración

Aplicar cambios

migrate actualiza la base de datos

Comandos esenciales:

```
# Detectar cambios en modelos y crear archivos de migración
python manage.py makemigrations

# Aplicar las migraciones a la base de datos
python manage.py migrate

# Ver el estado de las migraciones
python manage.py showmigrations

# Ver el SQL que generará una migración
python manage.py sqlmigrate auth 0001
```

Este proceso garantiza que todos los desarrolladores del equipo tengan la misma estructura de base de datos y que los cambios se apliquen de manera consistente en todos los entornos.

Migraciones para Auth

Cuando creas un proyecto Django nuevo, las tablas del sistema de autenticación se crean automáticamente durante la primera migración. Esto incluye todas las estructuras necesarias para usuarios, grupos y permisos.

```
# Al crear un proyecto nuevo, ejecuta:
python manage.py migrate

# Para migrar específicamente el sistema auth:
python manage.py migrate auth

# Para ver las tablas creadas:
python manage.py dbshell
.tables
```

auth_user Información básica de usuarios: username, email, password hash, fechas	**auth_group** Grupos de usuarios con permisos comunes: editores, moderadores, etc.
auth_permission Permisos específicos: add_post, change_user, delete_comment	**auth_user_groups** Tabla de relación muchos-a-muchos entre usuarios y grupos

Estas tablas forman la columna vertebral del sistema de autenticación de Django y están optimizadas para manejar millones de usuarios con rendimiento eficiente.

Actividad práctica 2

💻 Desafío guiado

01

Crear proyecto nuevo

Inicia un proyecto Django desde cero:

```
django-admin startproject mi_auth_app
cd mi_auth_app
```

02

Migrar tablas de auth

Aplica las migraciones iniciales:

```
python manage.py migrate
```

03

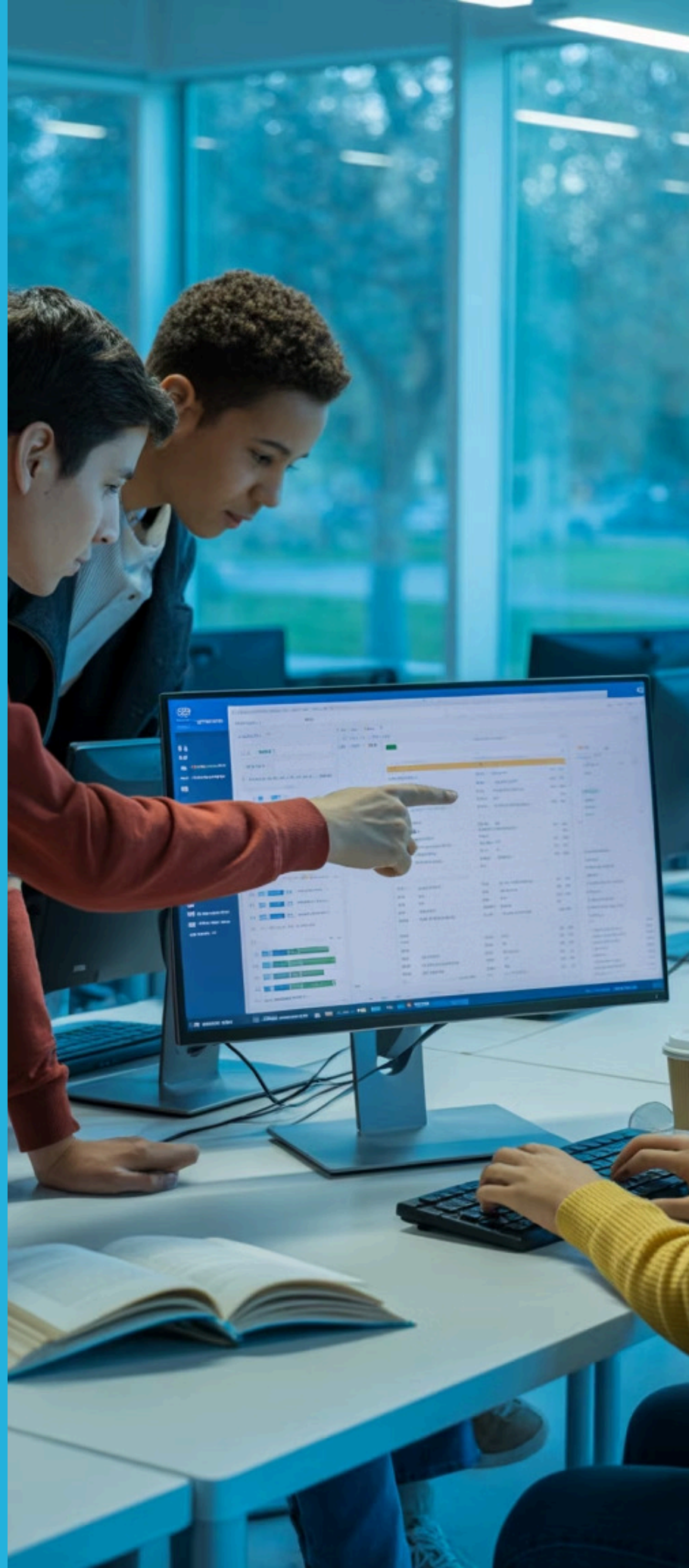
Explorar la base de datos

Utiliza DB Browser for SQLite o el shell de Django para examinar la estructura:

```
python manage.py dbshell
.schema auth_user
```

> 👉 **Objetivo principal: Identificar la tabla auth_user y comprender su estructura. Observa cómo Django organiza los campos de usuario, las fechas de creación y los campos de estado.**

Esta exploración práctica te ayudará a entender cómo Django almacena la información de usuarios a nivel de base de datos, lo cual es fundamental para el desarrollo avanzado.



Autorización con permisos

Los permisos en Django permiten un control granular sobre qué acciones puede realizar cada usuario. Puedes asignar permisos específicos individualmente o a través de grupos.

```
from django.contrib.auth.models import Permission, User
from django.contrib.contenttypes.models import ContentType
from myapp.models import Post

# Obtener un permiso existente
perm = Permission.objects.get(codename="add_post")

# Asignar permiso a un usuario
user = User.objects.get(username="juan")
user.user_permissions.add(perm)

# Crear un permiso personalizado
content_type = ContentType.objects.get_for_model(Post)
permission = Permission.objects.create(
    codename='can_publish_post',
    name='Can publish post',
    content_type=content_type,
)

# Verificar si un usuario tiene un permiso
if user.has_perm('myapp.can_publish_post'):
    print("Usuario puede publicar posts")
```

View (Ver)

Permiso para visualizar contenido

Add (Agregar)

Permiso para crear nuevo contenido

Delete (Eliminar)

Permiso para borrar contenido

Change (Modificar)

Permiso para editar contenido existente

Autorización con grupos

Los grupos simplifican la gestión de permisos cuando tienes usuarios que necesitan el mismo conjunto de autorizaciones. En lugar de asignar permisos individualmente, creates grupos con permisos predefinidos.

```
from django.contrib.auth.models import Group, Permission, User

# Crear un nuevo grupo
editors = Group.objects.create(name="Editores")

# Agregar permisos al grupo
permisos = Permission.objects.filter(
    codename__in=['add_post', 'change_post', 'view_post']
)
editors.permissions.set(permisos)

# Asignar usuario al grupo
user = User.objects.get(username="juan")
user.groups.add(editors)

# Verificar si un usuario pertenece a un grupo
if user.groups.filter(name="Editores").exists():
    print("Juan es editor")
```

Editores

Pueden crear, modificar y ver contenido, pero no eliminarlo

Moderadores

Pueden revisar, aprobar y rechazar contenido de otros usuarios

Administradores

Acceso completo al sistema, incluyendo gestión de usuarios

Esta organización jerárquica facilita el mantenimiento y la escalabilidad de tu aplicación a medida que crece el número de usuarios.

Proyecto final de la clase

💻 Mini-proyecto práctico

Vamos a construir un sistema completo de autenticación que integre todos los conceptos aprendidos. Este proyecto te dará experiencia práctica con el flujo completo de usuarios.

Sistema de registro

Crear formulario de registro que permita a nuevos usuarios crear cuentas con validación de datos.

Funcionalidad de login

Implementar vista de inicio de sesión con manejo de errores y redirección inteligente.

Cierre de sesión

Agregar funcionalidad de logout segura con confirmación de cierre exitoso.

Menú personalizado

Mostrar diferentes opciones de menú según el estado de autenticación del usuario.

Vista restringida

Crear una página que solo usuarios logueados puedan ver usando @login_required.

Bonus: Agrega mensajes de feedback usando el framework de mensajes de Django para mejorar la experiencia del usuario.

Cierre y reflexión

✅ Hemos aprendido:

Fundamentos del Auth en Django

Comprendimos qué es el sistema de autenticación integrado y cómo proporciona seguridad robusta desde el primer momento.

Gestión completa de usuarios

Aprendimos a crear usuarios, manejar login y logout de manera segura, y gestionar sesiones efectivamente.

Migraciones y estructura

Exploramos cómo Django organiza los datos de autenticación en la base de datos y cómo aplicar migraciones.

Permisos y grupos avanzados

Dominamos el sistema de autorización granular que permite controlar exactamente qué puede hacer cada usuario.

> 👉
>
> **¿Cómo aplicarías Django Auth en un proyecto real, como una red social o un e-commerce? Piensa en los diferentes tipos de usuarios, permisos y funcionalidades que necesitarías implementar.**

Con estos conocimientos, estás preparado para implementar sistemas de autenticación robustos y escalables en tus proyectos Django. ¡El siguiente paso es ponerlo en práctica!