

Autenticación Avanzada en Django

Tabla de Permisos y Vistas de Autenticación

Hoy exploraremos los fundamentos de la seguridad en Django, incluyendo cómo funciona la tabla `auth_permissions`, las redirecciones de seguridad y las vistas de autenticación predefinidas. Todo con ejemplos prácticos que podrás implementar inmediatamente.

Pregunta inicial: 🙋 ¿Qué creen que significa "permisos" en una aplicación web?





Fundamentos de Permisos en Django

Reglas de Acceso

Los permisos son reglas que determinan qué puede o no puede hacer cada usuario en el sistema

Asociados a Modelos

Se vinculan a acciones específicas sobre modelos: agregar, modificar, eliminar y visualizar

Almacenamiento


Toda la información se guarda en la tabla `auth_permissions` de Django

Ejemplo real: Usuario "Cristian" → puede crear artículos pero no borrarlos. Esta separación de permisos permite un control granular sobre las funcionalidades del sistema.

La Tabla auth_permissions

Características Principales

- Django crea esta tabla automáticamente al ejecutar migraciones
- Almacena todos los permisos disponibles en el sistema
- Se actualiza automáticamente cuando defines nuevos modelos

 **Dato importante: Cada modelo Django incluye por defecto 4 permisos básicos: add_model, change_model, delete_model, view_model**



Explorando Permisos en Código

Ejemplo Práctico en VS Code

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

```
# Obtener todos los permisos
```

```
permissions = Permission.objects.all()
```

```
# Mostrar información de cada permiso
```

```
for permiso in permissions:
```

```
    print(f"Nombre: {permiso.name}")
```

```
    print(f"Código: {permiso.codename}")
```

```
    print("----")
```

Este código te permitirá visualizar todos los permisos registrados en tu aplicación Django. Ejecuta este fragmento en el shell de Django usando `python manage.py shell`.

Actividad: Ejecuta este código en tu entorno de desarrollo y comenta los resultados obtenidos.



Relación con Usuarios y Grupos



Usuarios Individuales

Asignación directa de permisos a usuarios específicos



Grupos de Usuarios

Permisos asignados a grupos completos (ideal para proyectos grandes)

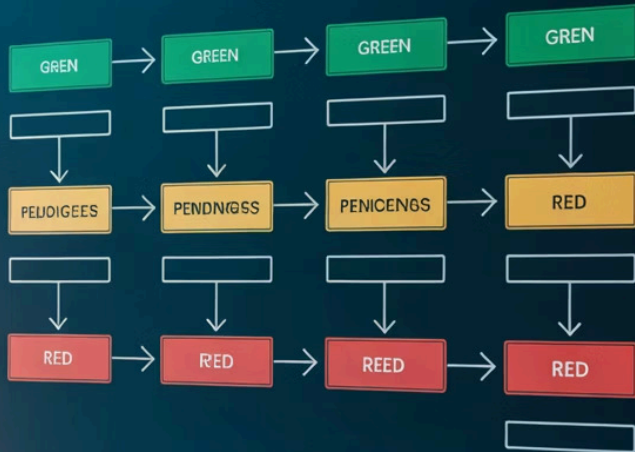


Control de Acceso

Sistema flexible que permite gestión escalable de permisos

Ejemplo práctico: Grupo "Moderadores" → permiso de `change_post` para editar publicaciones, mientras que usuarios regulares solo tienen `view_post`.

User Group Permissions



Redirigir Accesos No Autorizados

¿Qué Sucede Sin Autorización?

Cuando un usuario intenta acceder a una vista protegida sin los permisos adecuados, Django puede redirigirlo automáticamente a una página de login.

Configuración en settings.py:

```
# settings.py
LOGIN_URL = '/login/'
LOGIN_REDIRECT_URL = '/dashboard/'
```



Esta configuración asegura que los usuarios no autorizados sean dirigidos apropiadamente sin mostrar errores confusos.



Protegiendo Vistas con LoginRequiredMixin

Implementación Práctica

```
from django.contrib.auth.mixins import LoginRequiredMixin
from django.views.generic import TemplateView

class PanelUsuario(LoginRequiredMixin, TemplateView):
    template_name = 'panel.html'
    login_url = '/login/' # Opcional: sobrescribe LOGIN_URL
```

👉 Comportamiento: Si el usuario no está autenticado, se redirige automáticamente a /login/. Una vez autenticado, regresa a la vista original.

01

Usuario accede a vista protegida

El sistema verifica el estado de autenticación

02

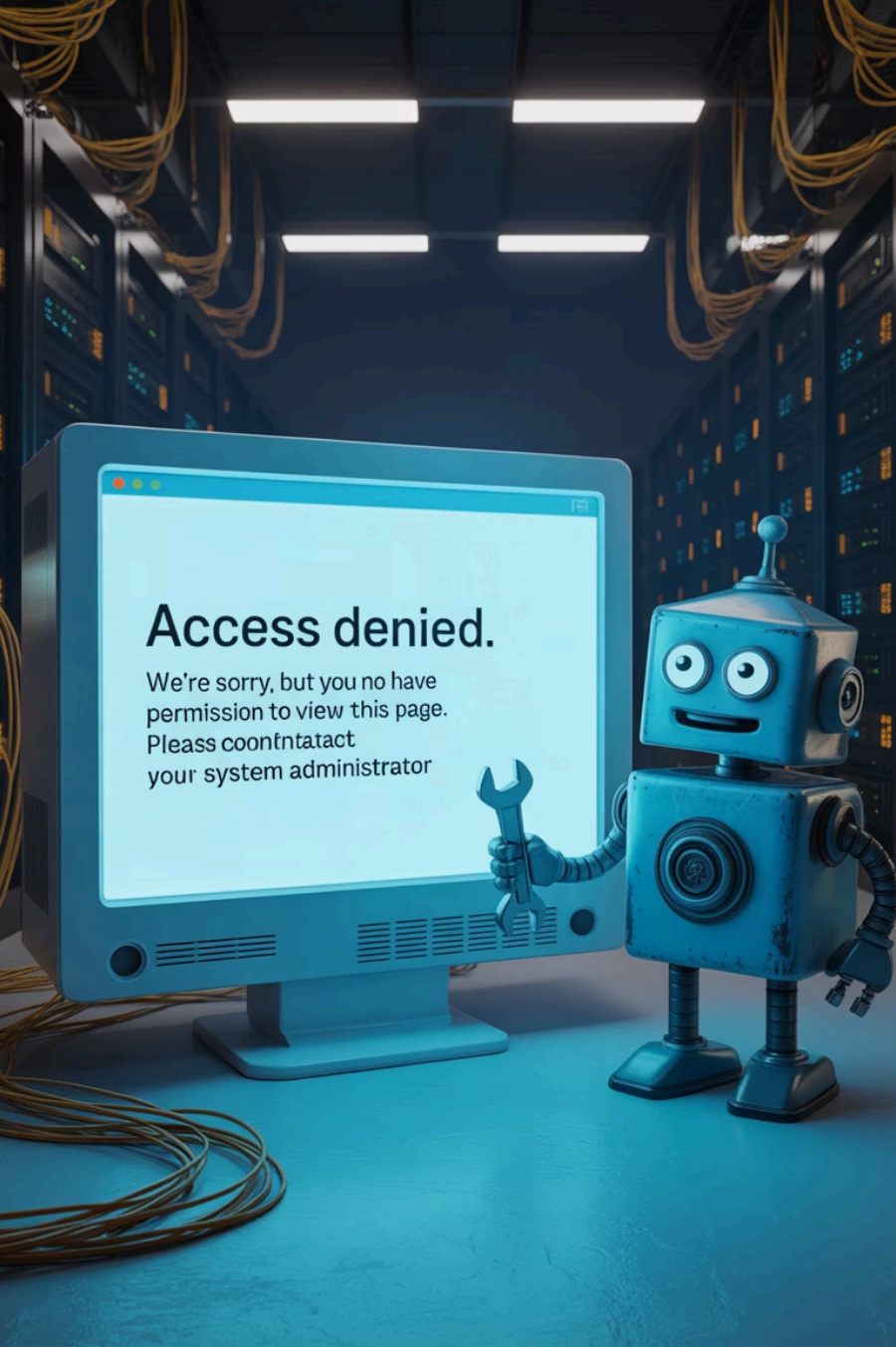
Redirección automática

Si no está autenticado, redirige a la página de login

03

Retorno a destino original

Después del login exitoso, regresa a la vista solicitada



Personalizando Redirecciones

Redirecciones Avanzadas

```
from django.shortcuts import redirect

class MiVistaProtegida(LoginRequiredMixin, TemplateView):
    template_name = 'panel.html'

    def handle_no_permission(self):
        # Redirección personalizada
        return redirect('pagina_de_error')
```

En lugar de enviar automáticamente al login, puedes crear redirecciones personalizadas que se adapten mejor a la experiencia de usuario de tu aplicación.

Actividad práctica: Crear una vista llamada `error.html` con un mensaje personalizado y configurar la redirección hacia esta página.

Vistas de Autenticación en Django

Django incluye vistas predefinidas que facilitan la implementación de sistemas de autenticación completos sin escribir código desde cero.

Login



Login

[Forgot password?](#)

[Don't have an account? "SIGN UP"](#)

LoginView

Maneja el proceso de inicio de sesión con formularios automáticos

LogoutView

Gestiona el cierre de sesión y limpieza de cookies

PasswordChangeView

Permite a usuarios cambiar sus contraseñas de forma segura

```
from django.urls import path
from django.contrib.auth import views as auth_views

urlpatterns = [
    path('login/', auth_views.LoginView.as_view(), name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
]
```

Configurando LoginView

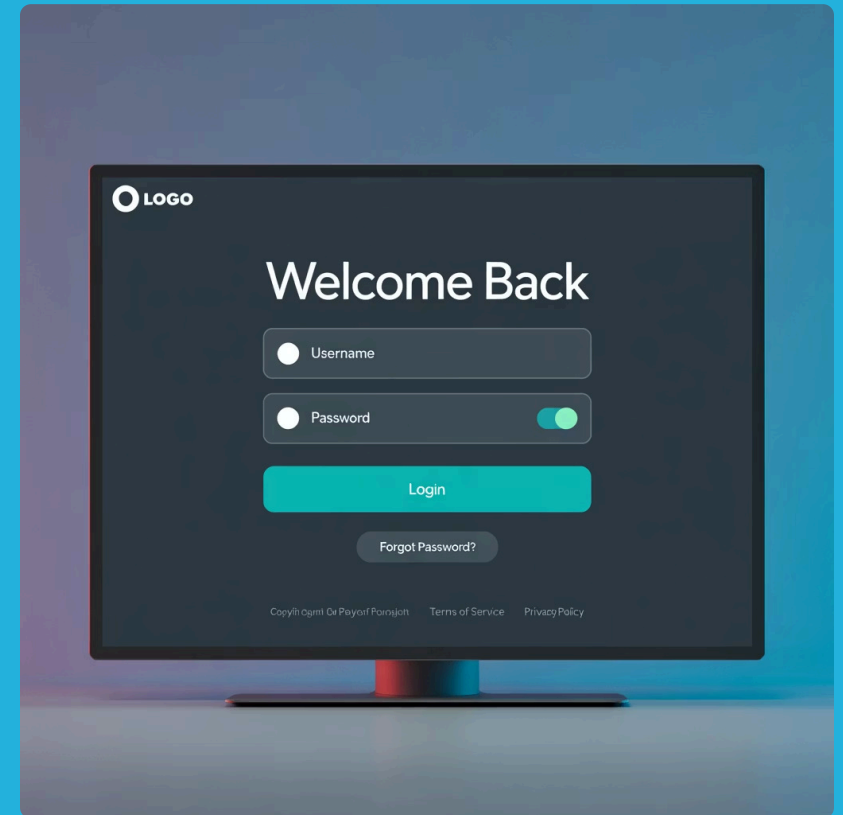
Funcionamiento de LoginView

LoginView muestra automáticamente un formulario con campos de usuario y contraseña. Puedes usar el template predeterminado de Django o crear uno personalizado.

Template personalizado (login.html):

```
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Iniciar Sesión</button>
</form>
```

Actividad práctica: Crear un template login.html con estilos CSS básicos para mejorar la apariencia del formulario.



Configurando LogoutView

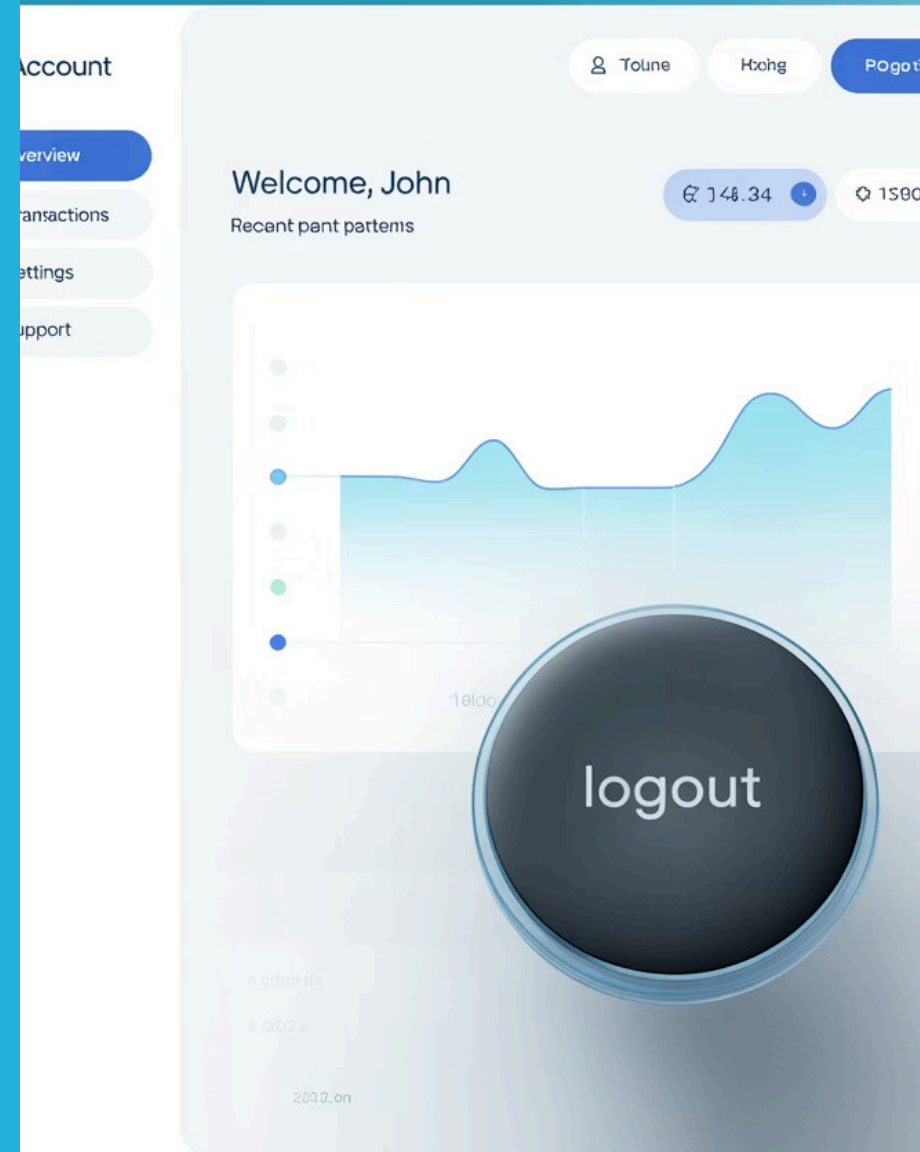
Gestión del Cierre de Sesión

LogoutView cierra la sesión del usuario, elimina las cookies de autenticación y redirige a una URL específica que puedes configurar.

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

# En urls.py
path('logout/',
     auth_views.LogoutView.as_view(next_page='home'),
     name='logout'),
```

- 1** — **Usuario hace clic en Logout**
Se activa la vista LogoutView
- 2** — **Limpieza de sesión**
Django elimina cookies y datos de sesión
- 3** — **Redirección**
Usuario es dirigido a la página especificada



Vista de Login Personalizada

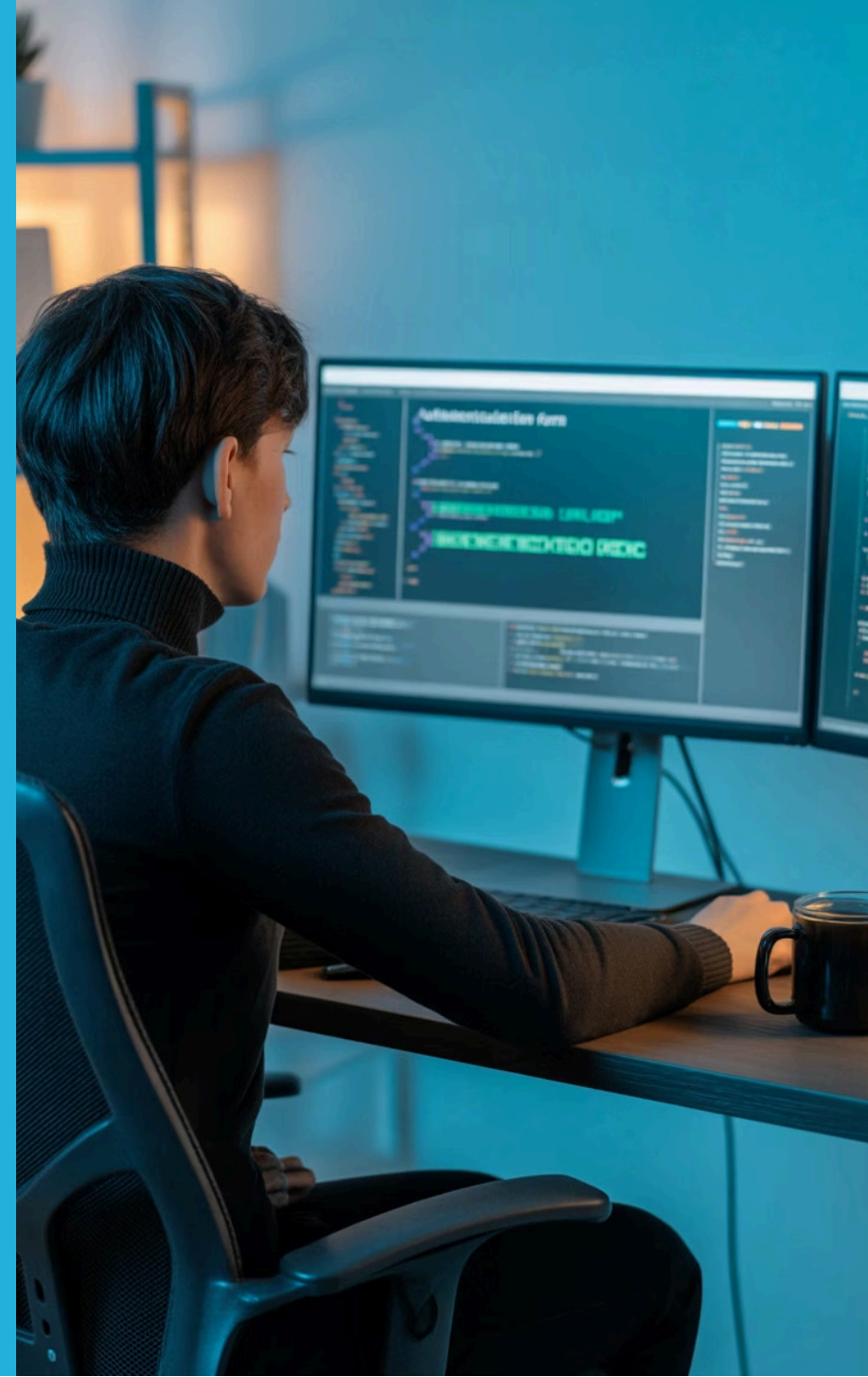
Creando Tu Propia Lógica de Autenticación

```
from django.contrib.auth.forms import AuthenticationForm
from django.shortcuts import render, redirect
from django.contrib.auth import login

def custom_login(request):
    if request.method == 'POST':
        form = AuthenticationForm(request, data=request.POST)
        if form.is_valid():
            user = form.get_user()
            login(request, user)
            return redirect('dashboard')
        else:
            form = AuthenticationForm()

    return render(request, 'custom_login.html', {'form': form})
```

Esta implementación personalizada te permite agregar lógica adicional como logging de intentos de login, validaciones extra, o mensajes personalizados de bienvenida.





Actividad Práctica en Parejas

¡Manos a la Obra!

1 Crear vista protegida

Implementar una vista llamada `panel_admin` que requiera autenticación

2 Configurar redirecciones

Redirigir usuarios no autenticados a `/login/` automáticamente

3 Template de error

Crear página de error personalizada para usuarios sin permisos

Tiempo estimado: 25 minutos. Trabajen en parejas y no duden en hacer preguntas durante el desarrollo.

Caso Práctico: Blog con Permisos

Sistema de Permisos Jerárquico

Implementaremos un blog con tres niveles de acceso diferenciados según el rol del usuario:



Administradores

Permiso `add_post`: pueden crear nuevos artículos y gestionar todo el contenido



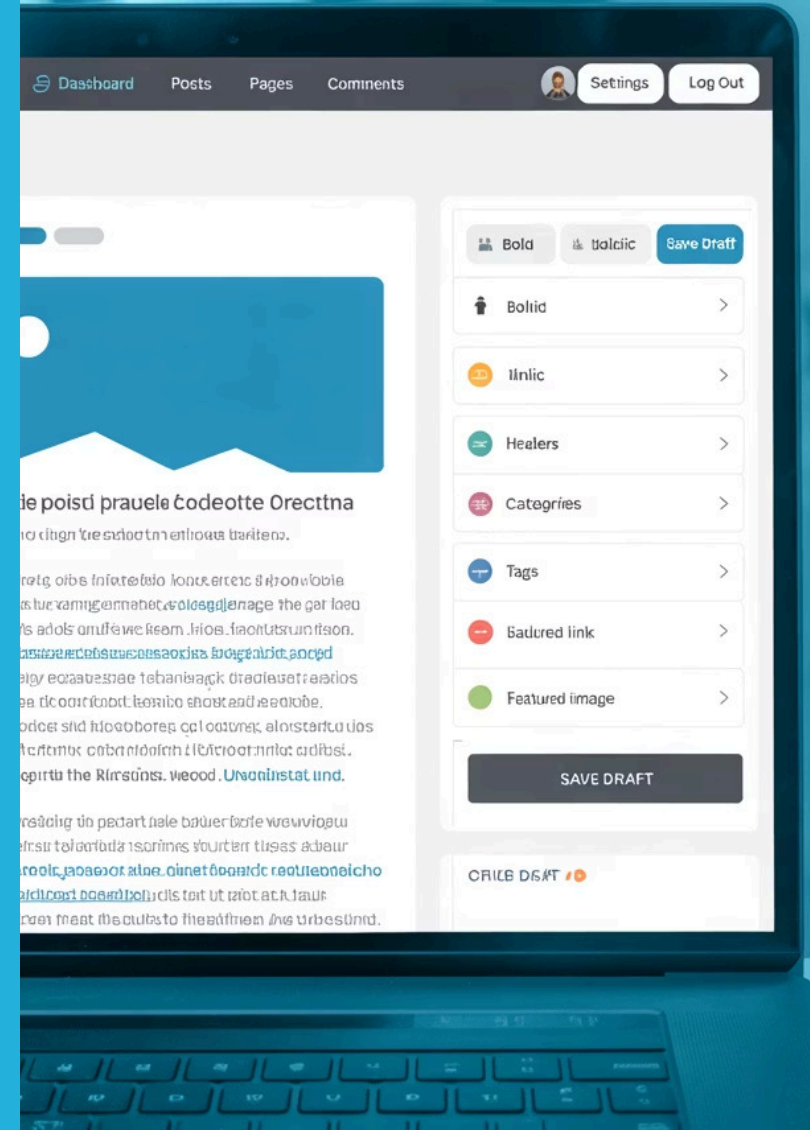
Moderadores

Permiso `change_post`: pueden editar artículos existentes pero no crear nuevos



Usuarios Registrados

Permiso `view_post`: solo pueden visualizar el contenido publicado



Consultando Permisos de Usuario

Verificación Programática de Permisos

En el shell de Django

```
user = User.objects.get(username='alex')
```

Verificar permiso específico

```
user.has_perm('blog.change_post') # Retorna True o False
```

Verificar múltiples permisos

```
user.has_perms(['blog.add_post', 'blog.delete_post'])
```

Obtener todos los permisos del usuario

```
user.get_all_permissions()
```

👉 **Pregunta reflexiva:** ¿Qué estrategias implementarías si el usuario no tiene el permiso necesario?



Asignando Permisos Programáticamente

Gestión Dinámica de Permisos

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

# Obtener el permiso específico
permiso = Permission.objects.get(codename='change_post')

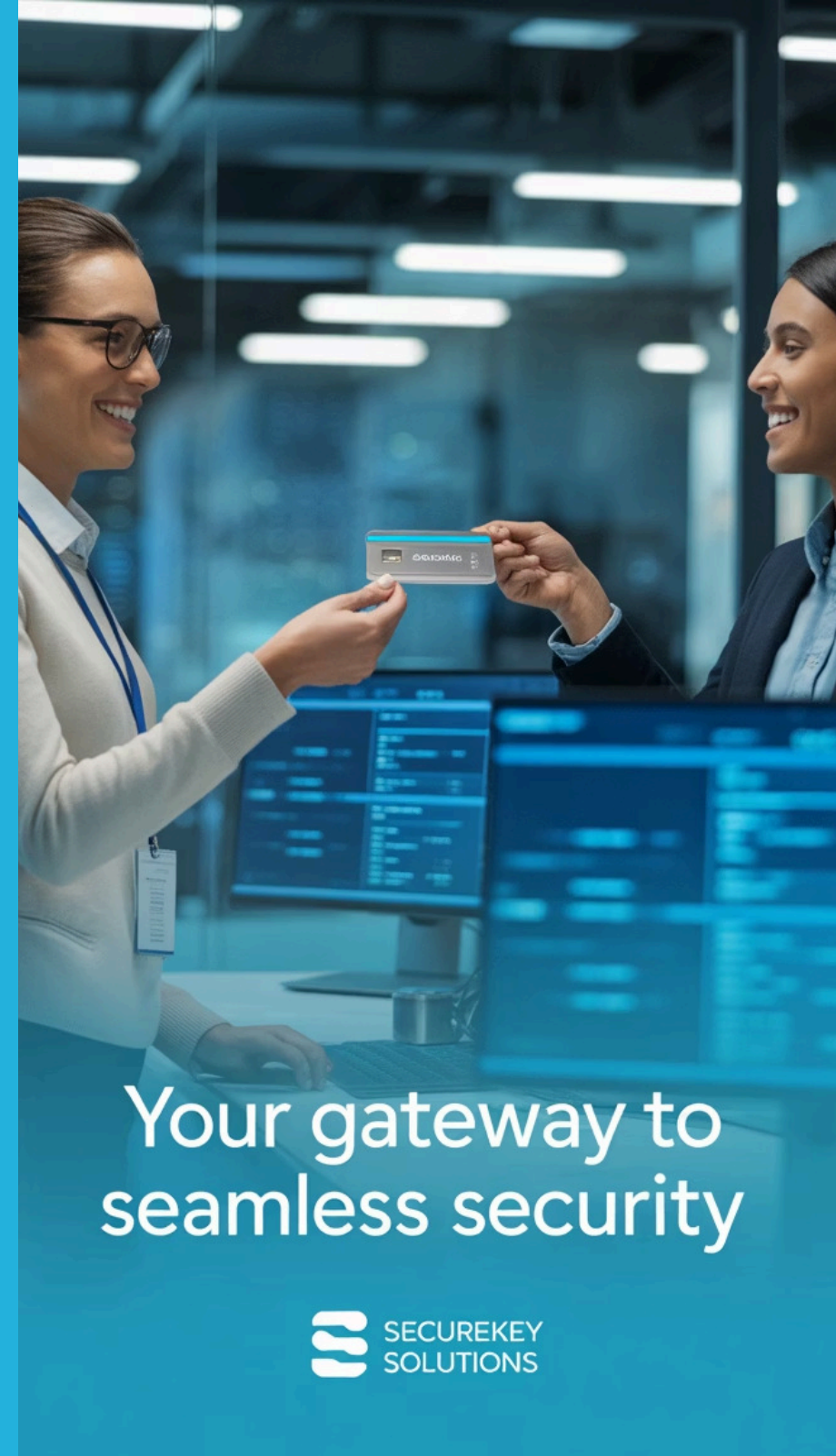
# Obtener el usuario
user = User.objects.get(username='alex')

# Asignar permiso al usuario
user.user_permissions.add(permiso)

# Remover permiso (si es necesario)
user.user_permissions.remove(permiso)

# Verificar cambios
print(user.has_perm('blog.change_post'))
```

Actividad práctica: Asigna un permiso específico a un usuario desde el shell de Django y verifica que los cambios se hayan aplicado correctamente.

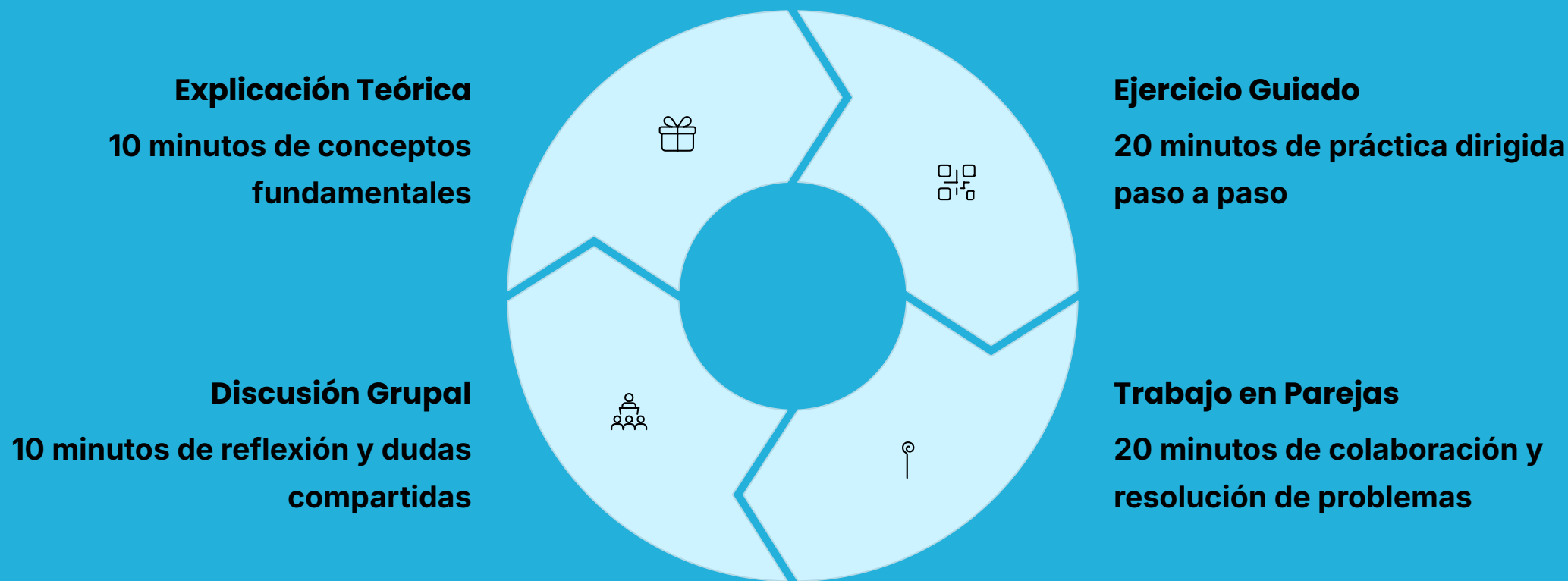


Your gateway to
seamless security



Metodología Activa de Aprendizaje

Nuestra sesión sigue una estructura diseñada para maximizar el aprendizaje práctico:



👉 **Filosofía:** Aprender haciendo, probando y equivocándose es la mejor forma de interiorizar estos conceptos.



Desafío Final: Sistema de Biblioteca



Reto

Implementa un sistema completo de permisos para una aplicación de biblioteca digital con roles diferenciados:



Administradores

- Agregar nuevos libros al catálogo
- Eliminar libros obsoletos
- Gestionar usuarios y permisos



Moderadores

- Editar información de libros existentes
- Actualizar disponibilidad
- Moderar reseñas de usuarios



Usuarios

- Visualizar catálogo de libros
- Realizar reservas
- Escribir reseñas

Resumen de Conceptos Clave

auth_permissions

Tabla que almacena todos los permisos del sistema Django automáticamente

Redirecciones

Control automático de accesos no autorizados mediante LOGIN_URL

Vistas de Autenticación

LoginView, LogoutView y otras vistas predefinidas personalizables

Usuarios y Grupos

Asignación flexible de permisos individuales o por grupos

Estos conceptos forman la base de cualquier sistema de seguridad robusto en Django. La comprensión de estos elementos te permitirá crear aplicaciones web seguras y escalables.



Reflexión y Próximos Pasos

¡Excelente Trabajo!

Para consolidar tu aprendizaje, reflexiona sobre estas preguntas fundamentales:

Importancia de la Seguridad

"¿Por qué es crucial manejar permisos correctamente en un sistema web moderno?"

Mejoras en la Aplicación

"¿Cómo mejora la seguridad de nuestra aplicación la implementación de estos conceptos?"

Aplicación Práctica

"¿Cómo aplicarías estos conocimientos en tu proyecto final del Bootcamp?"

La seguridad no es un añadido opcional, sino un componente fundamental que debe considerarse desde el diseño inicial de cualquier aplicación web profesional.

