Perfecto. Entonces vamos a crear un archivo llamado permisos.py que incluya:

1. Una función tiene\_permiso(usuario, accion)
2. Una estructura clara por rol
3. Posibilidad de extenderlo fácilmente luego a nivel de UI, JSON o tabla

**✅ ¿Qué controla esta función?**

| **Rol** | **Acciones permitidas** |
| --- | --- |
| Superadmin | Todas |
| Notario | crear\_documento, firmar, asignar\_procurador |
| Procurador | redactar, ver\_documento, editar\_borrador |
| Asistente | imprimir, generar\_aviso |
| AdminLocal | gestionar\_usuarios, ver\_estadisticas |

**🧠 Estructura propuesta**

# archivo: app/utils/permisos.py

# autor: Giancarlo F. + Tars-90

from app.models.enums import RolUsuarioEnum

# Mapa de roles → lista de acciones permitidas

PERMISOS\_POR\_ROL = {

RolUsuarioEnum.SUPERADMIN: ["\*"], # acceso total

RolUsuarioEnum.NOTARIO: [

"crear\_documento", "firmar", "asignar\_procurador", "ver\_documento"

],

RolUsuarioEnum.PROCURADOR: [

"redactar", "ver\_documento", "editar\_borrador"

],

RolUsuarioEnum.ASISTENTE: [

"imprimir", "generar\_aviso"

],

RolUsuarioEnum.ADMIN\_LOCAL: [

"gestionar\_usuarios", "ver\_estadisticas"

],

}

def tiene\_permiso(usuario, accion):

if usuario.rol not in PERMISOS\_POR\_ROL:

return False

acciones = PERMISOS\_POR\_ROL[usuario.rol]

return "\*" in acciones or accion in acciones

¿Te lo dejo listo para descargar como permisos.py en app/utils/?

Gracias por la explicación tan clara y detallada, Giancarlo. Con eso ya tenemos el panorama completo para modelar correctamente **las relaciones entre notarios, procuradores y asistentes**, tanto para bufetes pequeños como grandes.

**✅ Lo que haremos ahora**

Te voy a preparar un archivo Python llamado prueba\_relaciones.py, que podrás correr directamente con:

python scripts/prueba\_relaciones.py

Este archivo hará lo siguiente:

1. ✅ Buscar un notario y crear uno si no hay.
2. ✅ Buscar o crear uno o más procuradores para ese notario.
3. ✅ Mostrar cuántos procuradores tiene asignados el notario.
4. ✅ Mostrar si existe relación “inversa” (si el procurador trabaja con varios notarios).
5. 🔄 Dejar comentarios para modelar esto con una tabla intermedia muchos-a-muchos (notario\_procurador\_asociacion) si se decide implementarlo.

Te lo dejo listo en segundos.

Listo, Giancarlo. Aquí tienes el archivo de prueba para relaciones entre notarios y procuradores, listo para ejecutarlo y ver resultados reales desde tu base de datos:

**📘 Descargar prueba\_relaciones.py**

[Haz clic aquí para descargar](sandbox://mnt/data/prueba_relaciones.py)

**✅ Para correrlo:**

1. Abre terminal en tu entorno virtual
2. Ejecuta:

python scripts/prueba\_relaciones.py

Esto te mostrará:

* Si existe un notario
* Cuántos procuradores tiene el bufete
* Sugerencia para implementar relación muchos-a-muchos

Cuando lo ejecutes, dime qué ves y avanzamos con la tabla intermedia si quieres modelarla ya.

¡Gracias por la explicación tan clara, Giancarlo! Ahora sí vamos a hacerlo bien, desde el principio y para siempre 😎

**✅ Diagnóstico actual**

Tu carpeta static/bootstrap/ ya tiene **todo lo necesario localmente**:

* css/ con los estilos de Bootstrap
* js/ con los scripts necesarios

Esto **es lo correcto** y te evita:

* Problemas de red al usar CDN
* Conflictos por versiones
* Cambios de comportamiento inesperados

**🎯 Objetivo inmediato**

Te voy a entregar:

1. Un archivo base.html que:
   * Cargue Bootstrap desde local
   * Tenga navbar, contenedor, logo
   * Sea *heredable* por todas las plantillas
2. Una versión corregida de dashboard.html que extienda base.html
3. Todo con rutas absolutas que **no fallan jamás** si mantienes la estructura

**🗂️ Requisitos previos de carpetas:**

Por claridad, asegúrate de que existan:

app/

├── static/

│ ├── bootstrap/

│ │ ├── css/

│ │ └── js/

│ └── images/

├── templates/

│ └── base.html

│ └── dashboard/

│ └── dashboard.html

En static/images/ pon el logo (ej. logo\_bufete.png).

Listo, te preparo los archivos en un zip y los descargas. Un minuto.