¡Perfecto, Giancarlo! 🙌  
Aquí tienes el **prompt completo** para la **pestaña de Bloque 1 – Base de Datos y Seeds**.  
Cópialo y pégalo tal cual como **primer mensaje** en la pestaña 1:

**Prompt Inicial – Bloque 1: Base de Datos y Seeds**

Proyecto: Software Notarios | Bloque 1 - Base de Datos y Seeds

Usuario: Giancarlo Figueroa (pinedayfigueroa@gmail.com)

Contexto:

- Estamos trabajando en el desarrollo del software "Software Notarios"

- Objetivo de este bloque: Inicialización completa de la base de datos de desarrollo con migraciones y seed inicial

Estructura actual del proyecto:

C:\Users\Usuario\Mi unidad\Software\_Notarios

├── app\

│ ├── models\

│ ├── dashboard\

│ ├── utils\

│ └── templates\

├── scripts\

│ ├── init\_dev\_db.py

│ ├── seed\_inicial.py

│ ├── test\_relaciones\_asociacion.py

│ └── (otros scripts de pruebas)

├── migrations\ (se genera dinámicamente)

├── errores\_bdd\_flask\ (logs de errores)

└── init\_dev\_db.bat (antes old\_init\_dev\_db.bat)

Alcance del Bloque 1:

1. Confirmar que `init\_dev\_db.bat` funciona correctamente para reiniciar la DB de desarrollo.

2. Ejecutar `seed\_inicial.py` para:

- Crear los roles de usuario (Superadmin, Admin Local, Notario, Procurador, Asistente)

- Crear bufete principal y notario principal

- Crear Superadmin

- Insertar cláusulas y puntos iniciales en `ClausulasPuntos`

3. Verificar con `flask shell` y/o scripts de prueba que:

- Los roles existen

- Hay un bufete con su plan asignado

- El notario y superadmin están creados correctamente

4. Generar `README diario` con:

- Avances

- Errores y soluciones

- Fragmentos de código importantes

- Pendientes para el siguiente día

Notas importantes:

- Toda la base de datos se maneja en PostgreSQL (UTF8, template0)

- Se usan `flask db init`, `flask db migrate` y `flask db upgrade` para manejar migraciones

- La carpeta `migrations` se regenera en cada Big Bang

- Los errores deben registrarse en `errores\_bdd\_flask\\*.log`

- Queremos que todo quede sólido para futuras modificaciones de modelos y enums

Objetivo final de este bloque:

- DB inicializada sin errores

- Migraciones aplicadas

- Datos base cargados con éxito

- Test de relaciones (`test\_relaciones\_asociacion.py`) pasando correctamente

**✅ Qué logramos con este primer prompt**

* Te reconoce como **Giancarlo Figueroa**
* Sabe que es **Software Notarios**
* Entiende que estamos en **Bloque 1**
* Tiene **todo el contexto y objetivos claros** para trabajar **sin lag**

Cuando lo pegues y envíes en la **pestaña 1**, avísame,  
y te preparo los **prompts iniciales completos para Bloques 2, 3 y 4** para llenar las otras pestañas. ¿Vamos así?

Proyecto: Software Notarios | Bloque 2 - Dashboard y Blueprints

Usuario: Giancarlo Figueroa (pinedayfigueroa@gmail.com)

Contexto:

- Desarrollo del software "Software Notarios"

- Objetivo: Crear un dashboard funcional y limpio, con navegación básica, usando Flask y Bootstrap local

- Ya contamos con:

- Base de datos inicializada

- Seeds con roles y bufete principal

- Carpeta app/dashboard/ con blueprint inicial

- Carpeta app/templates/ con base.html y dashboard/dashboard.html

Alcance del Bloque 2:

1. Configurar correctamente el blueprint `dashboard\_bp` en `app/dashboard`

2. Integrar `base.html` con:

- Navbar con logo desde `static/img`

- Enlaces a secciones (Dashboard, Usuarios, Documentos, Librería, etc.)

3. Crear `dashboard.html` con:

- Conteo de usuarios por rol

- Conteo de documentos por estado

- Espacios reservados para alertas/avisos

4. Verificar rutas:

- `/dashboard` funciona correctamente

- Navbar y assets (Bootstrap local, CSS, JS, imágenes) cargan sin errores

5. Generar `README diario` con:

- Capturas o descripciones de layout

- Código de blueprints y templates

- Pendientes visuales o de UX

Notas importantes:

- Usar Bootstrap local (`app/static/bootstrap/`)

- Evitar CDN para prevenir conflictos y errores visuales

- Mantener la estructura limpia: blueprints, templates, static

- Navbar debe quedar funcional y ser la base del resto de vistas

Proyecto: Software Notarios | Bloque 3 - Seguridad y Roles

Usuario: Giancarlo Figueroa (pinedayfigueroa@gmail.com)

Contexto:

- Software Notarios requiere control de acceso basado en roles

- Tenemos roles: Superadmin, Admin Local, Notario, Procurador, Asistente

- Objetivo: Autenticación y seguridad básica para la aplicación

Alcance del Bloque 3:

1. Implementar `Flask-Login` para:

- Login de usuarios con sesión

- Logout seguro

2. Crear decoradores o funciones para permisos:

- `@requires\_role(RolUsuarioEnum.SUPERADMIN)`

- `@requires\_roles([RolUsuarioEnum.NOTARIO, RolUsuarioEnum.PROCURADOR])`

3. Redirecciones post-login:

- Superadmin → Vista de administración general

- Notario → Dashboard principal de su bufete

- Otros roles → Vistas según permisos

4. Implementar seguridad mínima en Blueprints:

- Rutas de Dashboard y Documentos protegidas

- Redirección a login si el usuario no está autenticado

5. Generar `README diario` con:

- Implementación de Flask-Login

- Decoradores de permisos

- Pendientes de seguridad avanzada

Notas importantes:

- Roles definidos en `RolUsuarioEnum`

- Las rutas deben ser protegidas desde el inicio

- Mantener código modular: `app/utils/permisos.py` y blueprints

Proyecto: Software Notarios | Bloque 4 - Documentos y Librería

Usuario: Giancarlo Figueroa (pinedayfigueroa@gmail.com)

Contexto:

- La aplicación debe gestionar escrituras, actas y sus cláusulas/puntos

- Tenemos modelos iniciales y enums para cláusulas y puntos

- Objetivo: CRUD de documentos notariales completo y manejo de librería

Alcance del Bloque 4:

1. Crear y probar modelos para:

- Documentos notariales

- ClausulasPuntos (aplican a escrituras y actas)

- Relación documentos ↔ notarios/procuradores

2. Implementar CRUD básico:

- Crear documento

- Editar documento

- Eliminar documento (borrado lógico)

3. Integrar selección de cláusulas/puntos:

- Dropdown o buscador al crear documento

- Clasificación: General, Obligatoria, Sugerida, Personalizada

4. Conectar con librería del bufete:

- Guardar documentos

- Listar por estado (Borrador, Firmado, Archivado)

- Mostrar avisos o vencimientos

5. Generar `README diario` con:

- Tablas y modelos creados

- Pruebas CRUD

- Pendientes para integrar con dashboard

Notas importantes:

- Usar `borrado\_logico` para no perder historial

- Clasificar correctamente cláusulas y puntos

- Mantener relaciones consistentes entre notarios, procuradores y documentos

¡Entendido, Giancarlo! 🚀

Tomamos una **decisión estratégica**:

* La **documentación Sphinx** queda en estado **aceptable** para este bloque
* Más adelante podemos dedicar **un bloque exclusivo** para limpieza y optimización de docs
* Ahora **priorizamos ver la app funcionando y avanzar con los módulos**

**🔹 Estado actual de Bloque 1**

1. ✅ DB inicializada y migraciones aplicadas
2. ✅ Seed completado con usuarios y cláusulas
3. ✅ App probada en navegador (flask run)
4. ✅ README y documentación técnica generada (aunque con warnings)
5. ✅ Git con primer commit realizado

💡 **Bloque 1 se da por cerrado**.

**🔹 Siguiente paso: Bloque 2 (Dashboard & Blueprints)**

Para avanzar rápido:

1. **Estructura base de Blueprints y Templates**
2. app/
3. dashboard/
4. \_\_init\_\_.py
5. routes.py
6. templates/
7. base.html
8. dashboard/
9. index.html
10. partials/
11. navbar.html
12. footer.html
13. static/bootstrap/
14. **Primer dashboard funcional** accesible en /dashboard:
    * Navbar con logo
    * Conteo de usuarios y placeholders de documentos
    * Bootstrap local funcionando
15. **Validación en navegador** → Ya verías el **inicio de la interfaz real**.

¿Quieres que prepare **el árbol de Blueprints + archivos base** para arrancar **Bloque 2** de inmediato?  
Así en la próxima hora ya tendrías el **Dashboard inicial visible**.