**Solicitud para la Creación de un Sistema Notarial Completo y Seguro (Backend y Frontend)**

**Objetivo:** Generar el código fuente completo de un sistema de gestión notarial robusto, seguro y funcional, con un enfoque en la coherencia de la interfaz de usuario y la solidez del backend, basado en las lecciones aprendidas en un proyecto de desarrollo previo.

**1. Visión General del Sistema y Propósito**

El sistema es una aplicación web destinada a la **gestión integral de las operaciones de uno o varios bufetes jurídicos/notariales**. Su objetivo principal es automatizar, controlar y asegurar los procesos relacionados con clientes, bienes, documentos notariales y la administración interna. Es crucial que la aplicación sea de nivel profesional y alta confiabilidad.

**2. Stack Tecnológico Requerido**

* **Backend Framework:** Python 3.x, **Flask** (última versión estable).
* **Base de Datos:** **PostgreSQL** (con énfasis en su robustez, capacidades de transacciones y auditoría).
* **Manejo de ORM y Migraciones de DB:** **SQLAlchemy ORM** y **Flask-Migrate (Alembic)** para la gestión y evolución del esquema de la base de datos de manera controlada.
* **Autenticación de Usuarios:** **Flask-Login**.
* **Interfaz de Administración (Opcional pero Deseado):** **Flask-Admin** para una gestión ágil de los modelos de la base de datos, con integración de la seguridad por roles.
* **Frontend (HTML/CSS/JS):** HTML5, CSS (deseable **Bootstrap 5.3.x** para asegurar estabilidad, funcionalidad y una estética limpia y consistente), JavaScript (mínimo, solo para la interactividad de Bootstrap y funcionalidades específicas como dropdowns si Bootstrap lo requiere).
  + **Lección Aprendida CRÍTICA:** Se **desea evitar completamente Tailwind CSS** y sus complejidades de integración en el tiempo de desarrollo de la UI. Se prioriza la estabilidad, las convenciones y los componentes predefinidos de Bootstrap 5 para una implementación más eficiente y predecible de la interfaz.
* **Iconografía:** Bootstrap Icons (prevalente) y/o Font Awesome 6.x (para íconos específicos si Bootstrap Icons no los tiene).

**3. Requisitos Funcionales Clave**

* **3.1. Gestión Multi-Bufete (Multi-tenancy):**
  + La aplicación debe soportar múltiples bufetes jurídicos.
  + Cada usuario del sistema debe estar asociado a uno (o, en el caso del Super Administrador, tener acceso a todos) bufetes.
  + Debe garantizarse el **aislamiento total de datos** entre bufetes. Un usuario de un bufete NO debe poder ver ni manipular los datos de otro bufete. Esto se aplica a Clientes, Bienes, Cláusulas, Documentos, etc., mediante filtros por bufete\_juridico\_id en todas las consultas de base de datos.
  + Un usuario autenticado que solo tiene acceso a un bufete debe ser redirigido directamente al dashboard sin pasar por una pantalla de selección de bufete. Si tiene acceso a múltiples bufetes (solo Super Administrador), debe ver una pantalla de selección.
  + Feature flags, al crear el bufete, el superadmin, por medio de un click, check, señalará los módulos que tendrá ese bufete. Pero todos los bufetes tienen el core de la aplicación, o sea generar documentos notariales, eligiendo entre instrumentos públicos (formalidad extrema) con clausulas, y actas notariales con puntos, esto permitirá al Notario, una vez alimentado el sistema, entrar y buscar que tipo de documento va a generar:
    - Para ello elige la(s) persona(s), que pueden ser individuales o jurídicas,
    - Sobre los bienes que se hará el instrumento
* **3.2. Sistema de Roles y Permisos (Seguridad Crítica a Nivel de Backend y Frontend):**
  + **Roles Definidos (rol\_usuario\_sistema\_enum en PostgreSQL):**
    - Super Administrador: Control total sobre **todos los bufetes** y usuarios.
    - Administrador: Control total sobre los datos de **SU bufete asignado**.
    - Notario: Permisos específicos relacionados con la creación/firma de documentos de **SU bufete**.
    - Procurador: Permisos para gestiones de documentos y avisos de **SU bufete**.
    - Invitado: Permisos muy limitados (ej. solo ver ciertos listados de **SU bufete**).
  + **Control de Acceso Riguroso (Backend):**
    - Todas las rutas (@app.route, @blueprint.route) deben tener decoradores @login\_required y/o verificaciones de current\_user.rol para garantizar que solo los roles autorizados accedan.
    - En Flask-Admin (ModelViews y sus métodos is\_accessible(), can\_create(), can\_edit(), can\_delete(), on\_model\_change(), on\_model\_delete()):
      * Un Administrador solo puede crear, editar o eliminar usuarios y entidades de **SU propio bufete**.
      * Un Administrador **NUNCA** puede editar ni eliminar usuarios con el rol Super Administrador.
      * Un Administrador **NUNCA** puede cambiarse a sí mismo el rol o el bufete\_juridico\_id.
      * El Super Administrador tiene control ilimitado sobre todos los bufetes y usuarios.
      * Los campos sensibles (rol, bufete\_juridico\_id) en los formularios de edición de usuario deben ser readonly en el frontend si el usuario logueado no es Super Administrador.
  + **Registro de Usuarios:** **CRÍTICO: El registro público (/register) debe estar completamente DESHABILITADO.** Todos los usuarios del sistema (excepto el Super Administrador inicial) deben ser creados manualmente por un Super Administrador o Administrador (quien asignará el rol y el bufete).
* **3.3. Dashboard Principal (/dashboard):**
  + Debe ser la página de inicio post-login.
  + Visualmente atractivo y organizado (utilizando tarjetas/cards de Bootstrap para cada sección principal).
  + Proporcionar enlaces a las secciones principales (Gestión de Clientes, Gestión de Bienes, Instrumentos Públicos, etc.).
* **3.4. Gestión de Entidades Maestras (CRUD Completo con Interfaz Pulida):**
  + **Bufetes Jurídicos:** CRUD (solo para Super Administrador). Incluye campos para nombre del bufete, notario principal, colegiado, NIT, dirección, teléfono, email, copyright de la aplicación logo path, nombre de la aplicación. (son datos fijos, se asignan al crear los bufetes) y se despliegan en el login y navbar.
  + **Usuarios del Sistema:** CRUD (para Super Administrador y Administrador - con las restricciones de rol/bufete detalladas). Incluye nombre de usuario, contraseña (hash), rol, estado activo, datos de contacto, asociación con Persona o Notario (opcional).
  + **Notarios:** CRUD (para Super Administrador y Administrador). Incluye nombre completo, número de colegiado (único), estado activo del colegiado, firma electrónica, etc., con asignación a un bufete.
  + **Personas Naturales (Clientes):** CRUD (con filtros por bufete).
  + **Personas Jurídicas (Clientes):** CRUD (con filtros por bufete).
  + **Bienes (Inmuebles y Muebles):** CRUD (con herencia de modelos y filtros por bufete).
  + **Cláusulas (Biblioteca):** CRUD (con filtros por bufete), para plantillas de texto reutilizables.
  + **Documentos** (explicado en el punto siguiente)
  + **Entidades Receptoras:** CRUD (ej. Registros, SAT, etc.), para avisos.
* **3.5. Documentos (core de la aplicación)**
  + todo documento Notarial tiene la siguiente estructura de manera general:
    - introducción (lugar, fecha, personas, motivo) esto ya es casi que un formato
    - contenido: el objetivo principal del documento, aquí van las estipulaciones puestas en clausulas, o lo que se hace constar en puntos)
  + para hacer cada documento la aplicación debe apoyar al Notario, con lo siguiente:
    - se deben tener grabados para cada documento (por ejemplo contrato de compra venta)
      * obligaciones previas a realizarlo (pedir documentos de identificación, prueba de propiedad, estar al día en impuestos,etc.)
      * los fundamentos legales, es decir , se graban los artículos de ley que indican las obligaciones, y avisos
      * obligaciones posteriores (emitir copias a los clientes, testimonio al principal,dar los avisos a las entidades receptoras)
    - para cada tipo de documento, contrato, acta, mandatos, etc se necesita que el sistema capture y avise:
      * las clausulas / puntos (si es acta) estas al grabarse se eligen de un checklist: sugerida, obligatoria, etc. Y tienen un campo, para poner tags, en las cuales se indica que esa clausula, es y puede ser utilizada en x cantidad de instrumentos,
      * para cada una / uno, se debe capturar los fundamentos legales, artículos de ley, y fundamentos doctrinarios, de manera que cuando se vaya a hacer un nuevo documento, y se le indiquen las clausulas o puntos, pueda ver esas motivaciones y razones para su contenido,
    - el sistema debe apoyar de tal manera que el Notario pueda elegir, crear un documento desde cero, eligiendo de un listado que se le despliega (contrato -> arrendamiento, compra venta, servicios, mandatos, etc.). entonces le hace el formato básico, y le agrega las clausulas posibles, dándole opción para agregar otras clausulas, y finalmente, eso produce un documento en Word, que se descarga para que el usuario termine de trabajarlo,
    - los documentos que se están trabajando en el sistema deben llevar un control:
      * en desarrollo,
      * correcciones
      * terminado, se considera listo (al llegar a este punto, si el modulo de timbres y papel protocolo está incluido en ese bufete, debe avisar la hoja serie y número que corresponde de inicio, y capturar cuantas utilizó, darles de baja en el inventario, obviamente indicando el papel que se utilizó, controlar si empezaron /terminaron en el anverso o reverso de las hojas)
      * cuantos timbres de que denominación se utilizaron,
    - contar con una librería en la cual se agreguen los documentos (formatos, templates, machotes según le llamen en cada lugar) que permitan elegir, los que ya se han hecho, y los nuevos pasan a formar parte de esa librería, haciendo cada vez más sencillo el trabajo al Notario.
* **3.6 Avisos**
  + esto se maneja asignando a una persona del bufete (procurador) que presente el o los avisos, para ello el sistema debe decirle que tiene que incluir: fotocopia del documento original, ponerle timbres, copias de los documentos de identificación,
  + entidad a la cual se debe llevar,
  + (si tiene el modulo de timbres, debe controlar los timbres utilizados)
  + los avisos deben ser controlados, con fechas y horas :
    - aviso creado
    - aviso preparado con su documentación,
    - aviso entregado a la entidad,
    - aviso rechazado por \_\_\_
    - aviso vuelto a hacer
    - aviso finalizado
* **3.7 Timbres y Papel Protocolo**
  + **Gestión de Inventario:** Modelos y CRUD para Timbres (Fiscales, Notariales - con registro de compras por serie/denominación/rango y estado de uso) y Papel Protocolo (registro de compras por quinquenio/serie/rango, asignación a Notario, control de hojas usadas).
  + **Generación de Documentos Notariales:** (Escrituras Públicas, Actas Notariales, Avisos) con correlativos automatizados, uso de plantillas y cláusulas predefinidas.
  + **Librería de Documentos Existentes:** Gestión de archivos Word ya creados (solo metadatos, no el archivo binario en DB).
* **3.8. Funcionalidades Adicionales (Importantes para el Negocio):**
  + **Auditoría (Borrado Lógico):** En lugar de eliminación física de registros importantes, implementar "borrado lógico" para mantener un historial.
  + Un Notario tiene asignado uno o varios procuradores,
  + En ciertos bufetes, un procurador atiende a varios Notarios, por lo que se hará necesarios saber que está trabajando y para quién,
  + **Dashboard Dinámico:** Indicadores y avisos pendientes (ej. avisos notariales a vencer, inventario bajo, que Notario y que procurador(es) tiene(n) que trámites/avisos/documentos asignados).
  + **Ayuda al Usuario:** Sección o enlaces para acceder a prontuarios legales, guías de uso, etc.

**4. Requisitos Técnicos y de Implementación (Detalles Cruciales)**

* **4.1. Estructura del Proyecto:**
  + La aplicación Flask debe estar organizada como un **paquete Python principal (app/)** en la raíz del proyecto.
  + app/ contendrá \_\_init\_\_.py (que define create\_app), config.py, models.py, cli.py, y las subcarpetas admin/ (para vistas de Flask-Admin) y routes/ (para Blueprints).
  + Los directorios templates/ y static/ deben estar en la **raíz del proyecto** (hermanos de la carpeta app/).
  + **.flaskenv** en la raíz del proyecto, con FLASK\_APP=app.
* **4.2. Manejo de Formularios (WTForms):**
  + Utilizar **macros de Jinja2** (templates/macros/\_form\_macros.html) para renderizar los campos de formulario de forma consistente y estilizada (usando clases de Bootstrap 5).
  + El macro render\_field NO debe duplicar los labels o campos.
  + Las descripciones de los campos (field.description) deben mostrarse como small tags (<small class="form-text text-muted">).
* **4.3. Migraciones de Base de Datos:**
  + Alembic (flask db migrate, flask db upgrade) debe funcionar perfectamente.
  + El archivo migrations/env.py debe estar correctamente configurado para importar create\_app, db y Config desde el paquete app.
  + **Manejo de ENUMs:** Las adiciones de valores a los ENUMs de PostgreSQL (ej. 'Invitado' al rol\_usuario\_sistema\_enum) deben ser manejadas con sentencias ALTER TYPE ... ADD VALUE en migraciones manuales (si es necesario).
* **4.4. Comandos CLI Personalizados:**
  + Un comando flask seed-cli initial\_setup que, al ejecutarse una única vez, crea el primer BufeteJuridico y un usuario superadmin asociado, si no existen.
* **4.5. Interfaz de Usuario (UI/UX - Bootstrap 5):**
  + **Navbar:**
    - Debe ser un navbar de Bootstrap 5 (navbar-dark bg-primary).
    - Mostrar el nombre\_aplicacion del bufete activo o un valor por defecto.
    - Contener menús desplegables (Gestion, Documentos Notariales) y un menú de usuario (current\_user.nombre\_usuario).
    - **Visibilidad Condicional de Menús:** Los menús "Gestión" y "Documentos Notariales" deben ser visibles **SOLO SI el usuario está autenticado Y ha seleccionado un bufete\_activo\_id** (es decir, después de la pantalla de selección de bufete).
    - Los dropdowns deben funcionar con los mecanismos estándar de Bootstrap 5 (clic en dropdown-toggle).
    - La información del usuario y el bufete activo (current\_user.nombre\_usuario (Nombre del Bufete)) debe ser legible.
    - La opción "Cambiar Bufete" en el menú de usuario debe ser visible **SOLO si el current\_user.rol es Super Administrador**.
  + **Footer:** Mostrar el app\_copyright del bufete activo o un valor por defecto.
  + **Páginas de Listado (Tablas):**
    - Todas las tablas (listado de bufetes, usuarios, notarios, clientes, etc.) deben estar estilizadas con clases de Bootstrap 5 (table table-hover table-striped table-sm, thead table-light, card shadow-sm).
    - La columna "Acciones" en las tablas debe mostrar íconos de "lápiz" (editar) y "bote de basura" (eliminar) con sus respectivos enlaces, utilizando un macro reutilizable (\_list\_actions\_macros.html) y clases de Bootstrap (btn btn-outline-warning btn-sm, btn btn-outline-danger btn-sm).
  + **Páginas de Formulario:**
    - Todos los formularios (agregar/editar bufete, usuario, notario, cliente, etc.) deben ser renderizados utilizando el macro \_form\_macros.html para asegurar consistencia.
    - Deben tener títulos de sección (h1, h2, h3 con clases de Bootstrap para organización visual).
    - Los campos de formulario deben ser legibles y bien alineados.

**5. Lecciones Aprendidas (para evitar errores y optimizar el desarrollo)**

* **Evitar la Complejidad Innecesaria en UI:** **NO USAR Tailwind CSS** en esta fase del proyecto debido a los conflictos y el tiempo de desarrollo extra que generó en la interfaz. **Ceñirse estrictamente a Bootstrap 5** y sus convenciones para la UI/UX para mayor estabilidad y eficiencia.
* **Gestión de Módulos Python:** Ser extremadamente riguroso con la **estructura de paquete (app/) y las importaciones relativas** (from .models import ..., from ..forms import ...). Asegurar que FLASK\_APP=app y que el \_\_init\_\_.py principal cree la instancia de la aplicación correctamente.
* **Manejo de ENUMs de PostgreSQL:** Cuando se añaden nuevos valores a un ENUM existente en la base de datos, no basta con actualizar el modelo Python. A menudo se requiere una migración manual (op.execute("ALTER TYPE ... ADD VALUE 'NuevoValor'")).
* **Seguridad por Roles Multicapa:** La seguridad no es solo un decorator. Debe ser validada en:
  1. La capa de ruta (@login\_required, if current\_user.rol in ...).
  2. La capa de Flask-Admin (is\_accessible, can\_create, can\_edit, can\_delete).
  3. La capa del formulario (haciendo campos readonly o disabled para ciertos roles).
  4. La capa de lógica de negocio (validación final en on\_model\_change/on\_model\_delete para impedir acciones no autorizadas).
* **Context Processors para Datos Globales:** Usar context\_processors (lambda functions) es la forma más limpia de inyectar datos (como bufete\_info) que necesitan estar disponibles en todas las plantillas.
* **Plantillas Reutilizables:** Maximizar el uso de macros Jinja2 (\_form\_macros.html, \_list\_actions\_macros.html) para la renderización de campos y acciones, asegurando consistencia y reduciendo código duplicado en las plantillas individuales.
* **Limpieza de Caché:** La importancia de instruir al usuario final sobre cómo limpiar el caché del navegador (Ctrl+F5) para ver los cambios CSS/JS/HTML.